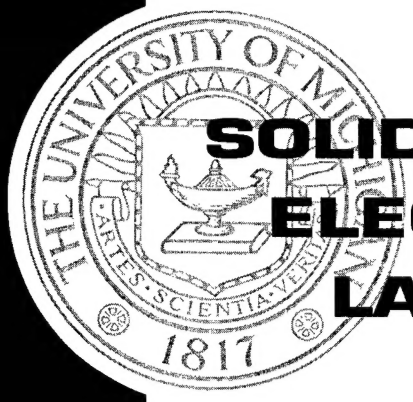
A QUANTITATIVE APPROACH TO NONLINEAR IC PROCESS DESIGN RULE SCALING

TECHNICAL REPORT NO. SSEL-288

1999

By

Spencer Montgomery Gold



SOLID-STATE ELECTRONICS LABORATORY

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
THE UNIVERSITY OF MICHIGAN, ANN ARBOR

This report has also been submitted as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the University of Michigan, 1999.

SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1999	Technical

4. TITLE AND SUBTITLE
A Quantitative Approach to Nonlinear IC Process Design Rule Scaling

5. FUNDING NUMBERS
DAAH04-94-G-0327

6. AUTHOR(S)
Spencer Montgomery Gold

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
University of Michigan
Department of Electrical Engineering
1301 Beal Ave.
Ann Arbor, MI 48109-2122

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 33790.73-EL

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis introduces a methodology for determining scaled horizontal process design rule values that reach an effective tradeoff between not only cost and area, but performance. This is accomplished with a procedure that iteratively finds the design rules that have the greatest impact on minimum layout area, and reduces them to their points of diminishing return from a cost, area, and performance perspective. The primary instrument for performing this analysis is a process-independent RAM compiler.

This thesis also describes optimization algorithms for exploring the large SRAM transistor size design space, and gives an innovative approach for optimizing an entire synchronous SRAM.

Finally, a cost/benefit analysis of CGaAs transistor threshold voltage scaling is described. Through PUMA RAM compiler-based performance evaluations and die cost estimations, it was shown that CGaAs transistor threshold voltage scaling is expensive (with respect to expected benefits), compared to horizontal design rule scaling.

14. SUBJECT TERMS			15. NUMBER IF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

Advances in integrated circuit processing technology have enabled an astonishing increase in microprocessor performance in recent years. Maximum operating frequencies continue to rise as transistors become smaller and faster. Meanwhile, increasing integration levels are expanding the possibilities for microarchitectural implementation. The impetus behind these improvements is the scaling of the IC manufacturing process. Process scaling is a multi-faceted effort which typically involves shrinking every physical design rule. These rules, which govern the minimum dimensions and spacing requirements for layout geometries, are a critical factor in determining the area required to implement a design, and thus directly affect power, delay, and cost. Design rule scaling plays a significant role in improving microprocessor performance by allowing a design to become smaller, faster, and more efficient.

1.1 Process Scaling Techniques

An examination of past IC process generations [1-3] reveals two major techniques that have been used to scale process design rules. The most common method, known as a linear shrink, has its roots in Constant field scaling proposed by Robert Dennard in 1973 [4,5]. In this approach, voltages and all vertical and horizontal dimensions are reduced by the scaling factor κ (where $\kappa > 1$), while doping densities are increased by the same factor, as shown in Table 1.1. Constant field scaling is advantageous because it preserves the strengths of all electric fields in the device as well as device transconductance and layout power density [6]. Because of the need to maintain consistent power supply voltages, process engineers have employed Constant voltage scaling, which is also described in Table 1.1. Constant voltage scaling is similar to constant field scaling in also reducing all vertical and horizontal dimensions by the scaling factor κ, while increasing the doping density by the same factor. The supply voltage, however, is not scaled,

Parameter	Scaling Model	
	Constant field	Constant voltage
Length (L)	$1/\kappa$	$1/\kappa$
Width (W)	$1/\kappa$	$1/\kappa$
Supply Voltage (V)	$1/\kappa$	1
Substrate doping (N_A)	κ	κ
Gate oxide thickness (t_{ox})	$1/\kappa$	$1/\kappa$
Transconductance (g_m)	1	κ
Junction depth (X_j)	$1/\kappa$	$1/\kappa$
Electric field strength (E)	1	κ
Load Capacitance (C)	$1/\kappa$	$1/\kappa$
Power Density	1	κ^3

Table 1.1 Constant field and constant voltage scaling.

resulting in an increase of the transistor transconductance, electric field strengths, and power density.

Traditional constant field and constant voltage scaling involves the proportional reduction of all horizontal process design rules. Linearly scaling all horizontal design rules has been popular because it allows easy design migration from one process generation to the next. While advantageous from a CAD perspective, linear scaling presents numerous challenges to the process engineer that must be overcome, despite the cost. This method does not consider the performance advantages or costs associated with reducing the design rules. Moreover, geometries have been reduced dramatically in recent years, and it is not clear that the ratios established generations ago will be optimal for deep submicron designs [7].

The other common scaling methodology is the hybrid shrink. Hybrid processes bridge major process generations by linearly shrinking horizontal transistor design rules while using the metallization from the current generation. This method is less advantageous from a CAD perspective, but allows earlier availability of advanced processes. However, the cost effectiveness of this approach is again questionable because it fails to examine the performance benefits and costs of individual design rule shrinks.

Linear and hybrid design rule scaling have played an important role in allowing semiconductor manufacturers to achieve profitability [8]. Linear design rule scaling, for example, enhances performance at the expense of higher costs. These cost increases can be justified if the accompanying area and performance improvement is large. Hybrid scaling is less costly than linear scaling and can still produce significant area and performance improvements while reducing time-to-market, but hybrid processes are only partially scaled. In the deep submicron era, however, manufacturing equipment and development expenses are soaring [10-12]. Under these conditions, achieving performance improvements in the most cost effective manner becomes crucial.

In this thesis, a methodology is proposed for performing cost effective nonlinear design rule scaling. In nonlinear scaling, horizontal design rules can be reduced by different factors. This will allow an IC process to evolve in the directions with the greatest cost-performance advantages. While this technique will put greater demand on tools for layout migration, recent CAD tool developments (e.g., MasterPort, Lace, QuickPort, etc.) make practical the migration of designs to nonlinearly scaled processes.

Past design rule analysis and scaling methodologies have concentrated on discovering the relations between area and cost. The nonlinear design rule scaling methodology presented in this thesis is the first to include the impact on overall performance that design rule scaling provides.

1.2 Design for Manufacturability

The concept of design for manufacturability (DFM) [8,13-15] was pioneered by Wojciech Maly at Carnegie Mellon University and includes concepts, principles, and techniques for "profit maximization achieved via optimization of IC cost-performance tradeoffs." [8, pg. 692] Maly concludes that the focus of DFM is the "maximization of manufacturing volume achievable for lowest possible cost and with marketing-department-given performance." [8, pg. 692]

DFM, however, is only a related work and does not include methodologies and techniques for performing cost effective process shrinks. Current DFM methodologies center around analyzing an already existing process. Based on the results of this analysis, process engineers can then make recommendations that certain design rules be adjusted in

order to increase yield. In contrast, this thesis describes a methodology for nonlinear process scaling that is based on the cost, performance, and area impact of scaling each design rule. This methodology assumes that research and development efforts will be conducted, and that capital equipment acquisitions will be made, in order to achieve a process with the new design rule values.

1.3 An Embedded RAM Perspective

The impact of reducing individual design rules, or sets of design rules, on overall IC performance and area will vary according to each IC's type, size, and structure. Therefore, the problem of achieving cost-performance-area optimality in a nonlinear design rule shrink should be approached according to the target application.

1.3.1 Microprocessor-based Design Rule Scaling

Determining the effect of individual design rule reductions on overall microprocessor performance and area is a formidable challenge. From a microprocessor viewpoint, overall performance and area is significantly affected by factors other than manufacturing process characteristics. For example, microarchitectural features influence the ratio of transistors used for logic versus RAM, and can vary widely from one microprocessor design to the next. Since the structure of logic blocks and RAM differ greatly, the effects of scaling individual design rules is likely to produce different results for each structure. If a set of design rule reductions produce more significant improvements in RAM performance than in logic blocks, a microprocessor with a larger proportion of its transistors dedicated to RAM will see a greater improvement in performance and area.

Cell placement and routing techniques also play a large role in determining overall microprocessor performance and delay. Efficient cell placement and routing will reduce the proportion of overall delay due to interconnect, and increase the proportion of delay attributable to logic gates. Scaling the interconnect design rules could therefore produce varying results from microprocessor to microprocessor, depending on the efficiency of its cell placement and routing.

Finally, obtaining overall microprocessor performance and area improvement data associated with individual design rule shrinks requires the processor's layout geometry,

which is usually not available until after VLSI circuit development is complete. If the process design rules are to be optimized for a future microprocessor design, an existing microprocessor layout must be used. The accuracy of design rule optimization results using such layout may be comprised if the layout does not accurately represent the future microprocessor architecture, cell placement, or routing techniques.

1.3.2 Embedded RAM-based Design Rule Scaling

High performance microprocessors are utilizing increasing amounts of embedded RAM. In fact, Hewlett-Packard's PA-8500 microprocessor employs over 90% of it's 140 million transistors as embedded RAM. As this trend continues, embedded RAM will play an increasingly important role in determining the overall processor cost and performance.

There are numerous benefits of approaching nonlinear design rule optimization from an embedded RAM perspective. First, overall RAM performance is tightly coupled to process factors due to its dense and regular structure. Microarchitectural decisions do not, for the most part, influence overall RAM performance. Set associativity [23-28], for example, is a major factor in determining the maximum operating frequency of a cache memory. But the circuitry used to implement set-associativity is peripheral to the actual RAM core and does not need to be included in performance analysis since it does not directly influence the performance of the RAM core itself.

Another distinct advantage of analyzing embedded RAMs is the ability to obtain accurate performance and area data with a single tool — a process-independent optimizing RAM compiler. This tool can generate compacted, optimized layout for a range of IC processes having different design rules. It can also make meaningful comparisons between IC processes and provides a way to collect the accurate performance and area improvement data necessary for cost, performance, and area tradeoff analysis.

Embedded RAM is good model for design rule cost/benefit evaluation. Moreover, the analysis methodologies described in this thesis can be directly applied to the evaluation of design rule scaling for logic or datapath blocks.

1.4 CGaAs as a Demonstration Technology

A description of Motorola's complementary GaAs (CGaAs™ [30-33]) IC manufacturing process was first published in 1993. This technology differs from E/D

MESFET processes by providing p-channel instead of depletion transistors. A cross-section of n- and p-channel CGaAs transistors is shown in Fig. 1.1. The CGaAs transistor channel is a buried layer of intrinsic InGaAs and the gate is formed by stacking layers of AlGaAs, GaAs, and TiWN to form a schottky diode with the InGaAs channel. This diode has a turn-on voltage of approximately 1.8 V, yielding devices that operate reliably within the 0.9 to 1.5 V range.

The availability of p-channel devices makes possible the use of complementary and dynamic, as well as source-coupled FET logic (SCFL) and direct-coupled FET logic (DCFL) circuits. CGaAs components can therefore be designed to have the performance of E/D MESFET components but with greatly reduced power dissipation. There are numerous advantages to using CGaAs for the implementation of high-frequency circuits, however, CGaAs has a maximum integration level that is lower than a typical CMOS process having comparable minimum geometries.

1.4.1 CGaAs Integrated Circuit Design

CGaAs closely resembles CMOS from a circuit design perspective. However, there are a few crucial differences between these technologies that make CGaAs circuit design unique. The CGaAs transistor structure differs from that of a MOSFET in that CGaAs gates will conduct current at voltages above 1.8 V, since their heterostructure insulated gates (HIGs) are essentially diodes. Even at levels below this turn-on voltage, CGaAs gates leak significant current.

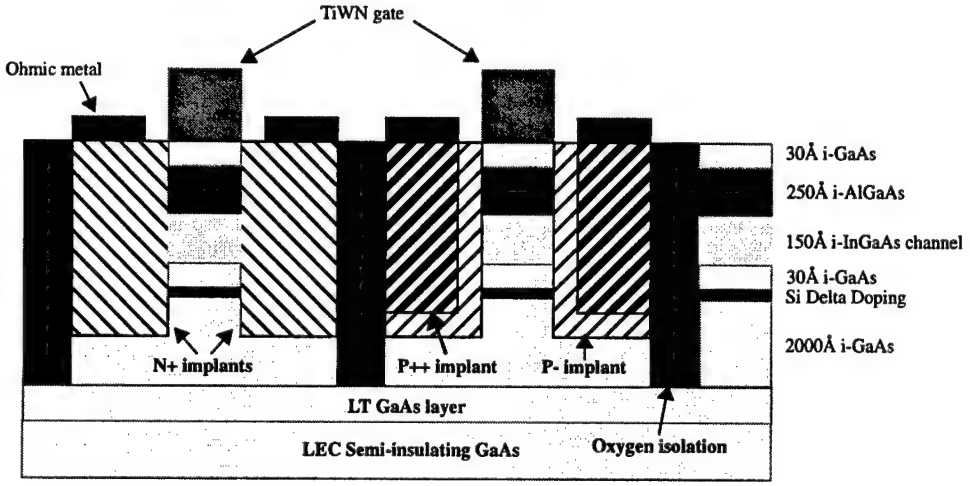


Fig. 1.1 CGaAs n- and p-channel transistor cross-section

A number of design rules are unique to CGaAs. P- and n-diffusions can directly abut and share a common contact. The large difference in charge carrier mobilities between n- and p-channel devices requires comparatively larger p-channel devices in CGaAs complementary gates to provide equal rise and fall times. Contacts to diffusion regions are made through an ohmic layer — an additional mask not included in CMOS processes. Furthermore, CGaAs' aluminum metallization allows only two vias to be stacked atop one another.

1.4.2 CGaAs Nonlinear Design Rule Scaling

There are several characteristics of CGaAs that make it a good candidate for nonlinear design rule scaling. First, CGaAs is a young technology with coarse design rules. When CGaAs was first introduced, n- and p-channel devices were implemented with 0.7 μm gate lengths and the other process design rules were fairly typical for a 0.7 μm process. In 1995, transistors with 0.5 μm gate lengths became available, but the remaining design rules were, with few exceptions, unchanged. Most of the original 0.7 μm design rules still exist in the current CGaAs process. CGaAs is therefore an excellent example of a process for which the horizontal design rules are not yet proportionally scaled. For this reason, CGaAs ICs stand to benefit greatly from a nonlinear shrink.

Another reason for choosing CGaAs as a demonstration technology is the availability of scaling cost estimates. With this data, meaningful and accurate results can be obtained from a nonlinear shrink cost/benefit analysis. It should also be noted that the techniques for performing CGaAs SRAM cost/benefit analysis can be applied to CMOS, or any other process technology. Moreover, the cost/benefit analysis methodology can be used in evaluating logic or datapath blocks.

CGaAs is the implementation technology for the PUMA high-performance microprocessor project [34]. The goal of this DARPA-funded research effort is to produce a radiation-hard microprocessor-based system utilizing advanced packaging techniques. An essential part of developing the PUMA microprocessor included producing optimized, embedded RAMs that serve as primary instruction and data cache memories. The process-independent optimizing RAM compiler developed for performing nonlinear design rule scaling analysis has also been used to generate memories for the PUMA microprocessor.

1.5 Thesis Overview

The costs of scaling deep submicron process design rules are diverging. The increasing use of hybrid processes is evidence of this fact. Meanwhile, commercial layout migration tools are increasing in number and capability [35-39]. As design rule scaling costs continue to diverge while layout migration and compaction tools become more usable, nonlinear design rule scaling should become more common. Once a nonlinear scaling program has been initiated, a process is free to evolve in those directions that best enhance cost and performance.

Determining design rule reduction ratios that provide the most cost efficient increase in both performance and area is central to effective nonlinear scaling. This thesis describes a methodology and CAD tools for determining the RAM power, delay, and area improvements that can be achieved when a design rule, or set of design rules, is scaled. The scaling process begins by identifying the design rules that have the greatest impact on layout area. These rules are analyzed to find the improvement in area, delay and power that result as the design rules are reduced through a range of practical scale factors. Performance and area improvement data is then combined with research and development cost estimates to produce a cost/benefit ratio, a quantitative metric for describing the cost-performance-area advantages of specific design rule reductions. The slopes and inflection points of cost/benefit vs. scale factor plots can be used to guide the process engineer in selecting scale factors for the various design rules. This procedure, if based on accurate models and applied iteratively, defines a method for scaling a process in the most cost-effective way.

The procedures and quantitative metrics used in IC process design rule cost/benefit analysis will be described in Chapter 2. Process-independent RAM compilation, including layout generation and circuit optimization, is described in Chapter 3. This chapter also describes the structure and capabilities of the PUMA CGaAs RAM compiler. The design rule cost/benefit analysis methodology is demonstrated using Motorola's 0.5 μm CGaAs process in Chapter 4. Conclusions are made in Chapter 5.

CHAPTER 2

DESIGN RULE COST/BENEFIT ANALYSIS

Profitability in the semiconductor industry is becoming increasingly dependent on the impact of research, development, and manufacturing equipment costs. As deep submicron IC manufacturing facility costs continue to soar, methods for making effective cost-performance tradeoffs become necessary. Extensive research has been conducted in the area of manufacturing yield maximization and design for manufacturability. This work, however, does not address the problem of cost effective process scaling. Instead, it provides models and analysis techniques for understanding the impact that design rules and layout characteristics have on manufacturing yield. These models and techniques can be used by the process engineer in recommending modifications to an existing process to decrease the susceptibility of a design to process variations and spot defects [40-42].

Process scaling is an expensive endeavor that requires research, development, and capital equipment acquisitions in order to be successful. Nonlinear design rule scaling allows these expenses to be kept in check while achieving significant improvements in both performance and layout area. The goal of design rule cost/benefit analysis is to guide the process engineer in selecting design rule reduction ratios that reach an effective tradeoff between die cost, performance, and layout area. This is done by identifying which design rules, when reduced, provide the greatest improvements in performance and area without requiring excessive scaling-related research, development, and capital equipment expenses. Design rules that do not significantly impact performance or area, or that are problematic from a cost perspective should not to be reduced. On the other hand, those design rules that can be reduced without excessive costs and provide significant area and performance improvements should to be reduced as much as is practical and cost-effective. Thus, an effective cost-performance tradeoff is made that provides an improvement in circuit performance and area in a cost-effective manner.

This chapter describes the methodologies for accomplishing an effective cost-performance tradeoff in selecting nonlinear design rule reduction ratios. Section 2.1 describes current semiconductor economic trends that have been developing. These trends form the motivation for emphasizing the management of IC development and manufacturing costs. Section 2.2 gives an overview of design for manufacturability, a methodology for maximizing profitability through yield enhancement. A description of the methodologies and quantitative metrics used in nonlinear design rule cost/benefit analysis is given in section 2.3. Concluding remarks are given in section 2.4.

2.1 Semiconductor Economic Trends

In 1964, Gordon Moore predicted that maximum integration levels would double every year [9]. This rate of doubling slowed to 18 months in the late 1970's, but it has remained unchanged to this day [10]. His prediction is now regarded as law, and its fulfillment has produced an industry whose growth and prosperity is unrivaled. As integration levels have increased, the cost per transistor has rapidly declined. This makes possible the production of ICs with ever increasing functionality and performance without requiring a similar increase in price. It is the affordability of leading-edge semiconductor technology that allows large volumes of ICs to be sold, and the entire semiconductor industry to prosper.

Numerous and difficult challenges have been overcome to maintain the pace of advancement predicted by Moore's law. As transistor gate lengths approach 0.1 μm, even greater challenges await, including reducing subthreshold current in low threshold devices, controlling threshold voltages, pattern printing, and interconnect scaling [7]. As process scaling challenges become more difficult, the costs associated with overcoming them will increase. Manufacturing facility costs have been rising exponentially and have already exceeded one billion dollars for a single fabline [11,12,43,44]. Furthermore, increasing global competition has led to shrinking profit margins [45-50]. As manufacturing costs continue to rise while profit margins erode, remaining profitable in the semiconductor industry will depend on the ability to reduce time-to-market while making effective tradeoffs between cost and performance.

2.1.1 Profitability in The Deep Submicron Era

The economic conditions of the deep submicron era have significantly changed the manner in which profitability is achieved. The primary force behind these changes is the rising costs of manufacturing state-of-the-art ICs. In the past, profitability could be maximized by producing a design that achieved maximum performance. In an era of exponentially rising manufacturing facility costs, however, minimizing expenses also has a great effect on the bottom line. The high costs associated with aggressive design rule scaling can be better managed, while enjoying the benefits of area and performance increases, by scaling each design rule according to their ability to impact cost and performance.

The increasingly competitive environment in today's semiconductor market allows revenue streams to be maximized only when volume production is achieved as quickly as possible. Producing a manufacturable design that meets target performance levels and that can be produced in volume is the goal of design for manufacturability.

2.2 Design for Manufacturability

The concept of design for manufacturability, or "ease of fabrication" [13] includes strategies and methodologies for use throughout the development and manufacturing cycles to increase the manufacturability of VLSI circuits. DFM eliminates the barriers between development phases and emphasizes the importance of making careful cost-performance tradeoffs early in the design cycle. DFM also suggests that manufacturability is a design goal that is as important as achieving target performance levels.

DFM methodologies and analysis techniques concentrate on optimizing an existing process for maximum yield. Process scaling is not considered a part of DFM, but is a precursor to it. Cost effective nonlinear scaling, however, can enhance the effectiveness of DFM by providing an initial process starting point that is more likely to produce a manufacturable design. DFM is therefore an important related work that is not superseded, but complemented by nonlinear scaling and design rule cost/benefit analysis. Because of its complementary nature, DFM goals, methodologies, and analysis techniques are summarized in the following sections.

2.2.1 DFM Goals

The overall objective of design for manufacturability is profitability, obtained by rapidly achieving volume production of high performance ICs. This objective is accomplished in two steps: 1) achieving non-zero initial yield, and 2) rapidly improving manufacturing yield [13]. Achieving non-zero initial yield is a nontrivial task and requires success in several different areas. First, process design rules must be chosen such that an effective compromise is made between minimum chip area and susceptibility to spot defects. Second, the sources of random environmental and process variations must be understood and characterized to an acceptable degree. This will enable the design of process-tolerant circuits. Third, a functionally correct design must be created that meets minimum performance levels. Finally, crucial equipment settings must be determined in order to successfully fabricate a functional IC.

Quickly ramping up the manufacturing process to produce high yields requires three capabilities. First, failure analysis must be performed correctly and efficiently. Second, process engineers and circuit designers must be able to predict how changes in equipment settings and design rules will impact performance. Finally, process parameters and IC layout must be efficiently modified (in the fewest number of iterations possible) to improve yield without compromising minimum performance objectives.

2.2.2 VLSI Process and Circuit Development Cycle

The development and manufacturing of a VLSI circuit can be divided into four phases, as illustrated in Fig. 2.1. These phases are planning, design, prototyping, and manufacturing. The planning phase can be viewed as the most crucial because the decisions made during this phase will determine the success or failure of the three subsequent phases. During the planning phase, the target application and a minimum performance level is defined. The implementation technology is then chosen with a set of layout design rules.

Upon completing the planning phase, the development cycle enters the design phase. Microarchitectural definition, logic and circuit design, functional verification, layout floorplanning, and layout verification are performed during this phase. If, during the design process, it is determined that the design will not meet target performance levels,

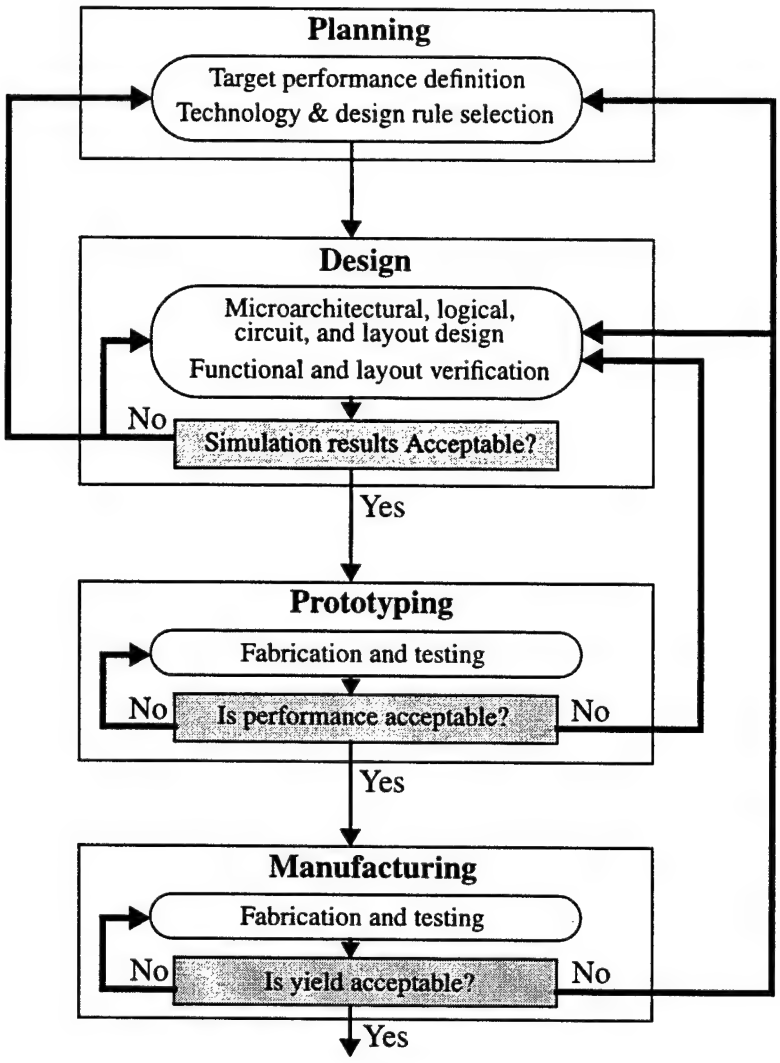


Fig. 2.1 The four phases of VLSI circuit development.

it may be necessary to return to the planning phase and make adjustments to the product definition or process technology.

After the IC has been designed and verified, the prototyping phase begins. The first design implementations are created using a pilot fabrication line. If the design does not operate correctly, or does not meet minimum acceptable performance levels, it becomes necessary to either tune the manufacturing process or return to the design phase and make adjustments to the IC layout. Once it has been demonstrated that the design can be fabricated to produce acceptable performance levels, the volume manufacturing phase begins.

During the manufacturing phase, process parameters are tuned to increase yield. Yield may depend, however, on other factors besides equipment settings or environmental parameters. Achieving high yield may require returning to the planning phase to adjust process design rules; or it may be necessary to return to the design phase to make the IC layout less susceptible to certain process variations or disturbances.

As indicated by Maly, there are two inherent factors that strongly influence the efficiency of the VLSI development cycle: the ability to predict yield and performance based on incomplete data, and the affects of random process disturbances [13]. The uncertainty associated with each of these factors lead to bad decisions during circuit development. Multiple iterations are then required through the development cycle to correct the effects of these bad decisions. With each iteration, data becomes more accurate and reliable, leading to better and more informed decisions, but crucial time and resources are spent. The efficiency of the VLSI development cycle, therefore, depends directly upon the number of iterations taken through each phase, which is dependent upon the level of uncertainty that exists in the decision making steps of each development phase. By eliminating the uncertainty associated with predicting performance and yield and by understanding the sources of random process and environmental disturbances, the VLSI development cycle becomes more efficient.

2.2.3 Choice of Design Rules

The planning phase of VLSI circuit development includes the selection of the implementation technology and layout design rules. Design rule optimization algorithms and tools have been developed to accomplish this purpose to some degree [14]. Razdan and Strojwas' Statistical Design Rule Developer (STRUDEL) determines values for each process design rule that maximizes yield while minimizing chip area [51]. STRUDEL uses statistical design rules, which are geometric design rules with accompanying probabilities of failure. This tool approaches the design rule optimization problem from a nonlinear scaling approach (i.e., each design rule can be reduced by a different proportion). However, chip area is obtained by estimation instead of by generating and measuring actual chip layouts. STRUDEL's capabilities were later expanded to create the Realistic Yield Evaluator (RYE), which used an iterative algorithm to produce design rule optimization results for hierarchical VLSI macrocells [52]. Neither STRUDEL nor RYE

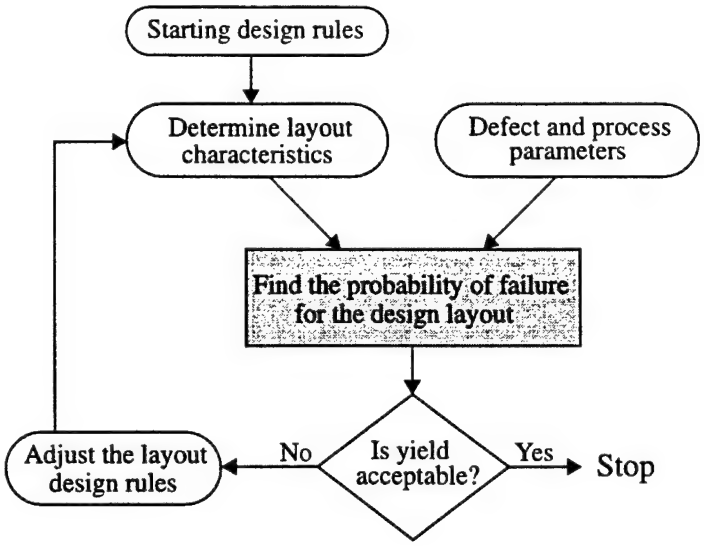


Fig. 2.2 RYE's iterative design rule optimization algorithm.

included performance, or equipment and development cost factors in their optimization algorithms.

RYE's iterative optimization algorithm, illustrated in Fig. 2.2, has become a standard technique for yield-based design rule evaluation [14]. Several other advancements have been made in yield modeling and forecasting. These advancements include: accounting for both global disturbances (line width variations, misalignment, lateral diffusions, etc.) and local disturbances (spot defects including missing or extra material, pinholes, junction current leakage, metal spikes, etc.) [52]; modeling yield losses in terms of defect size and layout characteristics instead of layout area alone [41,53-56]; considering the affects of both flat and 3-D spot defects [8,13]; accounting for data-dependent or cross-talk bridging faults [21,57]; relating design density to achievable yield [8,17]; and accounting for yield losses that are offset by functional redundancy [13]. As more comprehensive yield models are used, yield-based design rule optimization has become more useful and accurate.

Methodologies have also been developed that incorporate wafer costs into yield- and area-based design rule optimization [43,58,59]. These techniques are slightly more comprehensive than yield- and area-based approaches, but fail to include the costs associated with capital equipment acquisitions and research-related expenses necessary for design rule reduction.

Yield models that account for layout characteristics and density are likely to produce varying results for different layout structures. A design rule-based yield optimization technique that accounts for these differences is based on the concept of local design rules (LDRs) [60]. LDRs are useful for optimizing the yield of different types of layout structures by defining a set of aggressive global design rules (GDRs), then creating a set of LDRs for each type of layout structure. LDRs are then used to reduce the susceptibility of that structure to certain types of defects.

IC performance considerations are noticeably absent from current design rule optimization methodologies. This absence has not gone unnoticed. Maly indicates that the goal of DFM, "should be profit maximization achieved via optimization of IC cost-performance tradeoffs," and asks, "is such an optimization feasible right now?" His answer: "not really," due to the lack of adequate performance models and analysis tools [8, pg. 692]. Similarly, Strojwas concludes that design rule optimization algorithms need to, "take into account IC performance as well as reliability constraints." [14, pg. 458]

2.3 Performing a Design Rule Cost/Benefit Analysis

Cost effective nonlinear process scaling can be accomplished in three phases, as illustrated in Fig. 2.3. In the first phase, a cost/benefit analysis of the process design rules is performed. The results of this analysis guide the process engineer in selecting scale factors for each process design rule. Capital equipment acquisitions are made and the necessary scaling development efforts are conducted during the second phase of process development. Afterwards, the scaled process is available for IC manufacturing. During the third phase, DFM techniques are used to provide acceptable yield levels.

Design rule cost benefit analysis is an iterative procedure, as shown in Fig. 2.4. The procedure begins by determining a spending cap for scaling-related expenses, which

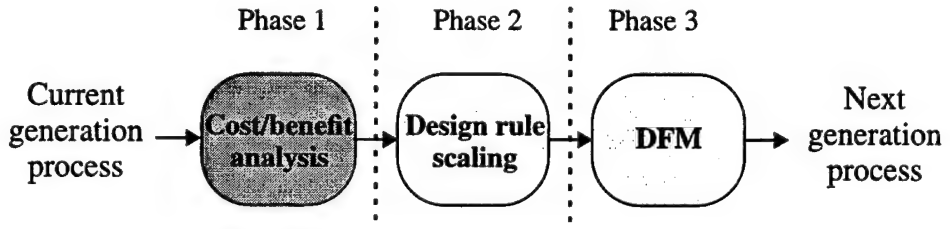


Fig. 2.3 Cost effective nonlinear process scaling.

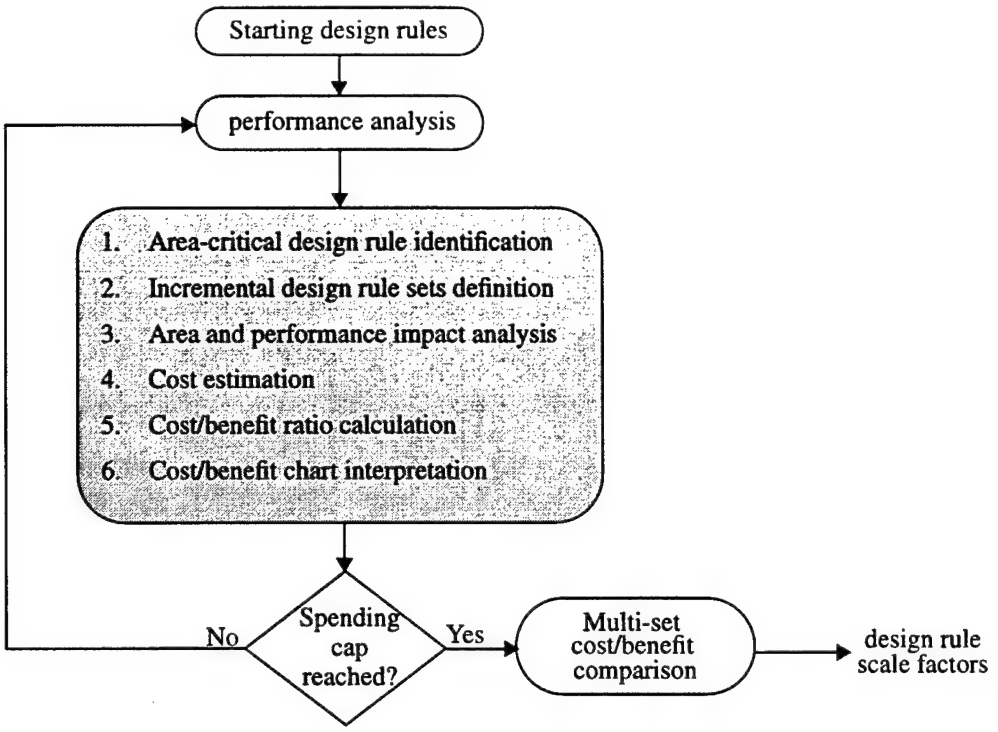


Fig. 2.4 Iterative cost/benefit analysis procedure.

will govern whether certain design rule sets and scale factors should be considered for reduction. A performance analysis of a representative design implemented with current generation design rules is then conducted. Next, the layout is examined to determine which design rules have the greatest impact on minimum area. Sets of area-critical design rules are formed and analyzed to quantify how scaling them impacts performance and area. The results of this analysis, illustrated in the form of a cost/benefit ratio vs. scale factor chart, guides the process engineer in selecting design rule scale factors. The scaled process becomes the baseline process for further cost/benefit analysis iterations. Once the process engineer has reached the predetermined spending cap, the results of each scaling step are compared to find the overall design rule reduction ratios that make the most effective tradeoffs between cost and performance. The following sections explain in greater detail how each step in this procedure is accomplished.

2.3.1 Identifying Area-Critical Design Rules

The first step in the iterative cost/benefit analysis procedure is to find the baseline performance and area of the representative design implemented in the current generation

process. These results are used for making comparisons to designs that are implemented in processes that have scaled design rules. Once this analysis is complete, the design rules that have the greatest impact on layout area are identified. Those rules that, when reduced, provide significant improvements in area are considered area-critical. Such rules are also likely to provide an increase in performance due to decreased parasitic capacitance and resistance. On the other hand, design rules that, when reduced, do not produce a significant improvement in area are not likely to affect performance and can be initially neglected. Once a set of area-critical design rules have been identified, it is beneficial to rank each rule according to its ability to impact layout area. This allows design rule sets to be created that will provide a reduction in layout area at minimal cost. The key lies in discovering which area-critical design rule set provides the most effective tradeoff between cost and performance.

Area-critical design rules can be scored and ranked according to the degree that they affect layout area. The area-critical score for design rule n is found by multiplying the number of occurrences (O_n) that the design rule is found in both the horizontal and vertical layout critical path by the design rule's value (V_n) in microns. A high score suggests that the design rule plays a significant role in determining minimum cell area. Fig. 2.5 illustrates this process for a single transistor layout.

For the layout in Fig. 2.5, the area-critical path in the horizontal (x) direction is determined by four design rules: active overlap of contact (aoc), contact width (cw), gate to contact spacing (gcs), and minimum gate length (gl). The vertical area-critical path is likewise constrained by six design rules: poly extension beyond active (pxa), active overlap of contact, minimum gate width (gw), contact width, poly to active spacing (pas), and poly overlap of contact (poc). Table 2.1 gives the values for O_n in both the x- and y-dimensions (O_{nx} and O_{ny}), O_n (the sum of O_{nx} and O_{ny}), V_n, and the corresponding area-critical score. The design rules are ranked according to their scores. As shown by the table, the contact width design rule is the most area-critical design rule.

Collecting the necessary data for identifying and ranking area-critical design rules can be an arduous exercise without the help of an automated layout analysis tool. Such

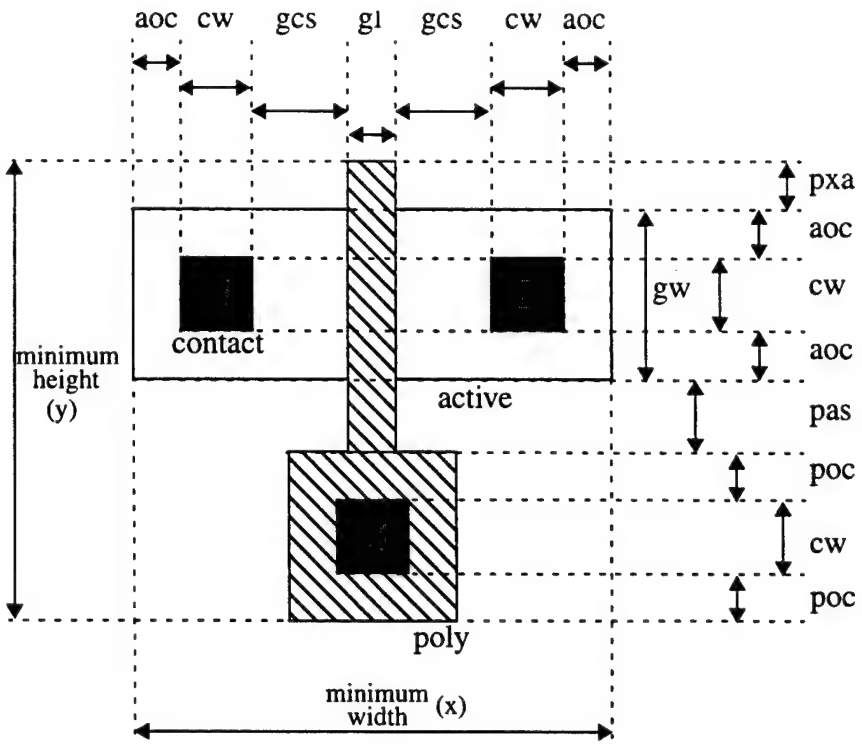


Fig. 2.5 Area-critical design rules for a single transistor.

tools, such as Duet Technology's MasterPort^{TM}, determines and graphically indicates the horizontal and vertical critical paths for an IC layout.

It is also important to discover the interdependencies between area-critical design rules. A dependent design rule is a rule that must be reduced simultaneously and proportionally with other design rules in order to produce an area improvement. During

n	Design rule	Symbol	O_{nx}	O_{ny}	O_n	V_n (μm)	Score
1	Contact width	cw	2	2	4	0.60	2.40
2	Gate to contact spacing	gcs	2	0	2	0.80	1.60
3	Minimum gate width	gw	0	1	1	1.20	1.20
4	Active overlap of contact	aoc	2	2	4	0.30	1.20
5	Poly to active spacing	pas	0	1	1	1.00	1.00
6	Poly overlap of contact	poc	0	2	2	0.40	0.80
7	Minimum gate length	gl	2	0	2	0.25	0.50
8	Poly extension beyond active	pxa	0	1	1	0.30	0.30

Table 2.1 Ranked area-critical design rules for a fictional process.

cost/benefit analysis, dependent design rules are combined with the design rules upon which they depend to form an interdependent design rule set.

Fig. 2.5 illustrates a dependent area-critical design rule. In the vertical critical path, the minimum width of the transistor is determined by the 1.20 μm minimum gate width design rule (gw). However, the minimum transistor width may not fall below the sum of the contact width (0.60 μm) and two active overlap of contact design rules (2 x 0.30 μm), which is also 1.20 μm. Therefore, the minimum transistor width rule depends upon the contact width and active overlap of contact width design rules. If the transistor minimum width is to be reduced, all three of these design rules must be simultaneously and proportionally reduced. Note, however, that the contact width and active overlap of contact width design rules are not likewise dependent upon the minimum transistor width design rule. Therefore, it is still beneficial to individually consider these rules for reduction.

2.3.2 Creating Incremental Design Rule Sets

With an understanding of which design rules are most likely to provide a significant reduction is layout area, it is possible to create combinations, or sets of design rules that, when reduced, provide a reduction in layout area of varying degrees. As previously mentioned, the objective of cost/benefit analysis is to guide the process engineer in selecting a set of design rule reduction ratios that: 1) provide a significant improvement in area and performance, and 2) provide this improvement in the most cost efficient way. By meeting these objectives, an effective tradeoff between cost and performance can be made. It is reasonable to conclude that the first of these objectives is best accomplished by reducing the area-critical design rules. The second objective can be met by scaling the area-critical design rules that provide a significant improvement in area and performance in the most cost efficient manner.

It would be beneficial to investigate every combination of the area-critical design rules to determine their impact on performance and area. However, for n area-critical design rules, 2^n-1 combinations exist. If a large number of area critical design rules are found, which is likely for a large design, the number of combinations is prohibitively large. There are two options for reducing the number of combinations that should be investigated without significantly compromising the accuracy of the results. First, the x

highest ranked area-critical design rules (where $x < n$) can be considered while neglecting all others. The total number of combinations then becomes 2^x-1.

The number of combinations to be examined can be further reduced without seriously compromising the results by considering two things: 1) the greatest amount of area improvement comes from reducing just the area-critical design rules, and 2) scaling costs can be lowered by reducing the fewest number of design rules. Therefore, the greatest reduction in layout area with the lowest associated cost is most likely to come from reducing the highest ranked area-critical design rule. This rule forms the first combination, or set of design rules to be examined. An incrementally greater reduction in area from the reduction of the fewest number of design rules is then most likely to come from reducing the two highest ranked area-critical design rules, and so forth. This pattern continues until the last area-critical design rule set, which includes the x highest ranked area-critical design rules, is created. Using this approach, x area-critical design rule sets can be investigated without compromising the accuracy of the results.

In the case of a dependent area-critical design rule, the dependent rule, together with the rules on which it depends, are included in the design rule set. Table 2.2 indicates the organization of seven area-critical design rule sets created from the layout of Fig. 2.5. The third design rule set includes both the minimum gate width and active overlap of contact design rules because the minimum gate width rule is dependent upon both the contact width and active overlap of contact design rules.

The seven design rule sets given in Table 2.2 represent the design rule combinations that are most likely to provide the greatest improvement in layout area. Because these design rule sets accomplish this area reduction by adjusting the fewest number of design rules, they are also most likely to provide the most cost efficient improvement in area and performance. This hypothesis is tested by performing an area and performance impact analysis of the area-critical design rule sets.

2.3.3 Area and Performance Impact Analysis

Area-critical design rules are identified and ranked in order to create design rule sets that are likely to provide an incremental improvement in area and performance with minimal cost. The degree to which each design rule set actually provides this

Design rule	Symbol	Rank	Design rule sets						
			1	2	3	4	5	6	7
Contact width	cw	1	X	X	X	X	X	X	X
Gate to contact spacing	gcs	2		X	X	X	X	X	X
Minimum gate width	gw	3			X	X	X	X	X
Active overlap of contact	aoc	4			X	X	X	X	X
Poly to active spacing	pas	5				X	X	X	X
Poly overlap of contact	poc	6					X	X	X
Minimum gate length	gl	7						X	X
Poly extension beyond active	pxa	8							X

Table 2.2 Area-critical design rule set organization.

improvement must then be determined. This involves implementing and optimizing the same design in several different processes, then making comparisons of the achievable performance levels and minimum area requirements in each process.

For process comparison purposes, it is inadequate to generate one optimized design in each process since it represents only a single design point in a large design space. To make a fair and comprehensive comparison of two processes, the optimal power-delay design space must be determined. This is not a trivial task since it involves finding a wide range of optimal power and delay values that can be achieved with a given set of layout design rules and device models. An optimal power-delay curve can be created by finding the minimized power dissipation level for a range of delay values. This curve represents the boundary between the achievable and non-achievable areas of the power-delay design space, as illustrated in Fig. 2.6.

The optimal performance of two processes can be compared by using their optimal power-delay curves. Fig. 2.7 illustrates how this comparison is accomplished. In this figure, the optimal power-delay curves for two processes are given. Process A is the baseline, or unmodified process, and process B is a modified version of process A. The average improvement in power ($\Delta power_{ave}$) provided by process B is the average difference in power dissipation between the optimal power-delay curves. Improvement in

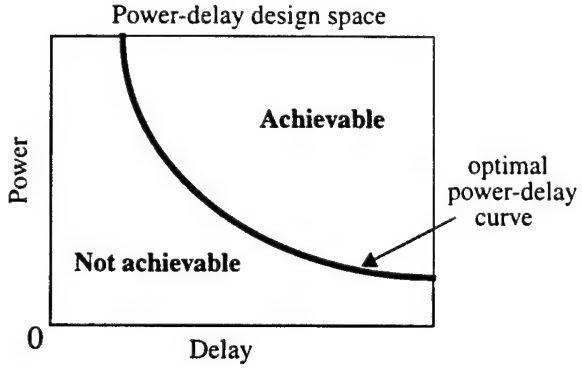


Fig. 2.6 An optimal design power-delay curve.

delay can be obtained either by finding the difference in minimum achievable delay ($\Delta delay_{min}$), or by finding the average decrease in delay ($\Delta delay_{ave}$) that is achieved for similar power dissipation levels.

If $P_n(d)$ represents the minimized power dissipation level that can be achieved in process *n* as a function of delay (d), and d_{max} and d_{min} represent upper and lower bounds on delay for the baseline process, $\Delta power_{ave}$ can be found using the following equation:

$$\Delta power_{ave} = \frac{1}{d_{max} - d_{min}} \int_{d_{min}}^{d_{max}} (P_A(d) - P_B(d))\, dd \qquad (2.1)$$

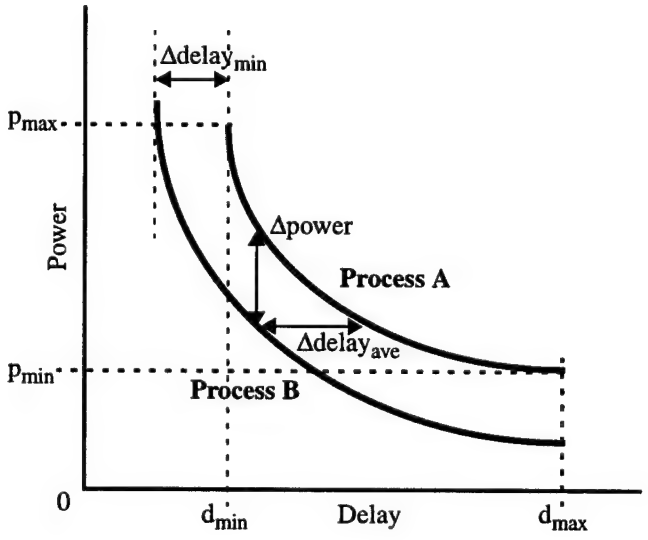


Fig. 2.7 Power and delay improvement from optimal power-delay curves.

If $D_n(p)$ represents the minimum achievable delay in process n as a function of power (p), and p_{max} and p_{min} represent upper and lower bounds on power dissipation for the baseline process, $\Delta delay_{min}$ and $\Delta delay_{ave}$ can be found using the following equations:

$$\Delta delay_{min} = \min(D_A(p)) - \min(D_B(p)) \qquad (2.2)$$

$$\Delta delay_{ave} = \frac{1}{p_{max} - p_{min}} \int_{p_{min}}^{p_{max}} (D_A(p) - D_B(p))\, dp \qquad (2.3)$$

Determining the area improvement that results when a set of design rules is reduced is accomplished in a similar manner. Each point along the optimal power-delay curve represents a design implementation with a measurable layout area. One method for calculating average area improvement ($\Delta area_{ave}$) involves finding the layout area for each point along the two optimal power-delay curves. This data can be used to form an optimal layout area versus delay curve, as illustrated in Fig. 2.8.

If A(d) represents the layout area required to implement a design having minimal power dissipation and delay d, and d_{max} and d_{min} represent the upper and lower bounds on delay for the baseline process, the average area improvement can be found by calculating the average area difference between design points of common delay, according to the following equation:

$$\Delta area_{ave} = \frac{1}{d_{max} - d_{min}} \int_{d_{min}}^{d_{max}} (A_A(d) - A_B(d))\, dd \qquad (2.4)$$

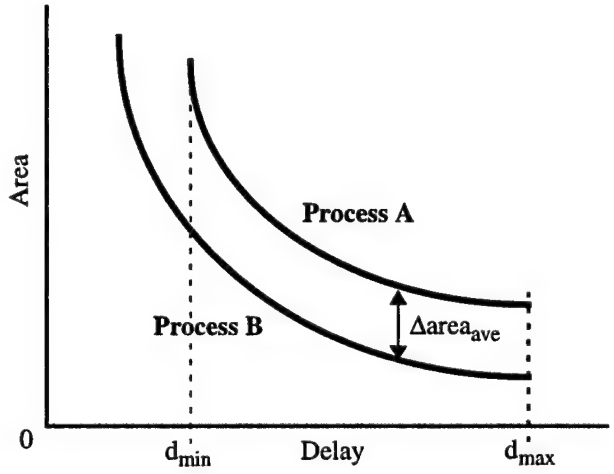


Fig. 2.8 Using optimal layout area curves to find area improvement.

Generating a design that has been optimized for a given process involves more than just compacting the original layout. Each transistor, signal wire, and power rail must also be appropriately resized. Moreover, creating an optimal power delay curve involves exploring the feasible design space by creating multiple optimized designs in each process. It is desirable that an automated approach be used in producing an optimal power-delay curve. A technique for generating these curves for SRAM implemented in any complementary technology will be given in chapter 3.

2.3.4 Cost Estimation

As stated earlier, the goal of cost/benefit analysis is to guide the process engineer in making effective cost-performance-area tradeoffs when selecting layout design rules. With an understanding of how area-critical design rule sets impact performance and area, it is necessary to find how reducing each design rule set through a range of practical reduction ratios will affect overall die cost. Die cost is a function of wafer cost, dies per wafer, and die yield, and can be calculated using the following equation [61]:

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per wafer} \cdot \text{Die yield}} \qquad (2.5)$$

The cost of manufacturing a single wafer (wafer cost) includes both starting material and fabrication expenses. Moreover, research and development expenses as well as capital equipment acquisitions made during the preliminary process scaling effort must also be included when determining wafer cost. The research and development expenses, and the depreciation of the manufacturing equipment, are amortized over the total number of wafers to be fabricated by the new facility, as indicated in the following equation:

$$\text{Wafer cost} = \frac{\text{R\&D cost} + \text{Depreciation}}{\text{Number of wafers}} + \text{Materials cost} + \text{Fab cost} \qquad (2.6)$$

The sum of the scaling research and development expenses and the capital equipment depreciation represent the total cost of scaling a process from one generation to the next. Since there is no way to exactly determine these costs beforehand, estimates must be created that use past expenses and projected equipment costs.

As a design rule, or set of design rules, are scaled through a range of proportions, the associated scaling expenses associated with research, development, and capital equipment acquisitions will increase. These expenses, which are amortized over the number of wafers to be fabricated, increase die cost. However, by scaling the design rules, overall die area decreases. This increases both the number of dies per wafer and die yield, and therefore reduces die cost. Therefore, the key in managing die cost is to perform an appropriate amount of scaling according to the number of wafers over which the scaling costs will be amortized. For high volume manufacturing facilities, extensive scaling efforts can be justified. But for low volume facilities, less scaling may be more effective. Certainly, there are other factors besides die cost that affect profitability. If, for example, a low volume process is not scaled aggressively in order to reduce die cost, the resulting performance levels may not be competitive. Design rule cost/benefit analysis results can be used to guide the process engineer in making effective tradeoffs between die cost and performance considerations.

Several models have been developed for predicting yield based on die area and lethal defect density [62-71]. Dingwall's equation [64], which is given as equation 2.7, is a model that relates the impact of die area and defect density on die yield.

$$\text{Die yield} = \left(1 + \frac{\text{Defect density} \cdot \text{Die area}}{c}\right)^{-c} \qquad (2.7)$$

In this equation, c is a parameter that represents the degree of clustering among the lethal defects. When c=1, the yield equation becomes Seed's formula [63], and when $c=\infty$ the equation becomes the Poisson yield formula [62]. A value of 3 or 4 is typically chosen for c when estimating yield for modern CMOS processes [61,72].

Defect density is a crucial variable upon which die yield depends. However, it is not possible to exactly predict the yield for a process that does not yet exist. Therefore, an implicit assumption must be made that defect densities will continue to follow the trends of past process generations. Once the newly scaled process becomes available, defect densities will probably be high. Defect densities, however, can be brought down to acceptable levels by following the techniques and using methodologies of DFM.

The number of dies per wafer can be calculated using equation 2.8 [72]. In this equation, the first term is the ratio of wafer area to die area. The second term approximates the number of die near the wafer periphery that are only partially fabricated by dividing the wafer circumference by the diagonal of a square die.

$$\text{Dies per wafer} = \frac{\pi \cdot (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \cdot \text{Wafer diameter}}{\sqrt{2 \cdot \text{Die area}}} \qquad (2.8)$$

2.3.5 Cost/Benefit Ratio Calculation

The cost benefit ratio should be appropriately defined for the target application. Wafer cost estimates should include capital equipment acquisitions and all associated development and production expenses necessary to reduce a set of design rules by a given proportion. The benefit portion of the cost/benefit ratio could be a product of power, delay, and area improvements, as in the following equation:

$$\text{Cost/benefit ratio} = \frac{\text{Die cost}}{\frac{P_b}{P_b - \Delta\text{power}} \cdot \frac{D_b}{D_b - \Delta\text{delay}}} \qquad (2.9)$$

In this equation, P_b and D_b represent the baseline power, delay, and layout area values. A cost/benefit ratio must be calculated for each reduction ratio of each area-critical design rule set. Once this data becomes available, it is possible to create cost/benefit vs. scale factor plots for each set of area-critical design rules.

2.3.6 Interpreting Cost/Benefit Plots

The slopes and inflection points of cost/benefit ratio plots can be used to guide the process engineer in selecting design rule reduction ratios that make an effective tradeoff in cost and performance. If the cost/benefit ratio is calculated for a given area-critical design rule set scaled through a range of practical reduction ratios, the scaling proportion that makes the most effective tradeoff from a pure cost/benefit standpoint is the point at which the cost/benefit ratio is smallest. The reduction ratio corresponding to the lowest cost/benefit ratio may also be described as the point of diminishing returns since scaling the design rule set beyond this point does not provide a more cost efficient improvement in performance and area.

Cost/benefit plots do not indicate how to scale a process in order to maximize overall profitability since the other factors upon which profitability depends are not considered in cost/benefit analysis. Instead, a cost/benefit plot indicates the point of diminishing returns for a particular set of design rules. In other words, the plot indicates how far a design rule should be scaled before it becomes necessary to reconsider all other design rules for reduction.

Fig. 2.9(a) and (b) show the die costs and benefit ratios of scaling a design rule by 10, 20, 30, and 40%. Fig. 2.9(c) illustrates the associated cost/benefit plot for the same scale factors. As indicated in Fig. 2.9(c), the point of diminishing returns corresponds to the reduction proportion with the smallest cost/benefit ratio, or 20%. Therefore, this design rule should be scaled by up to 20% before reconsidering all design rules for reduction.

Multiple area-critical design rule sets can be compared using cost/benefit ratio plots. The set of design rules that provides the most cost effective improvement in performance will have the lowest minimum cost/benefit ratio. Fig. 2.10 illustrates the

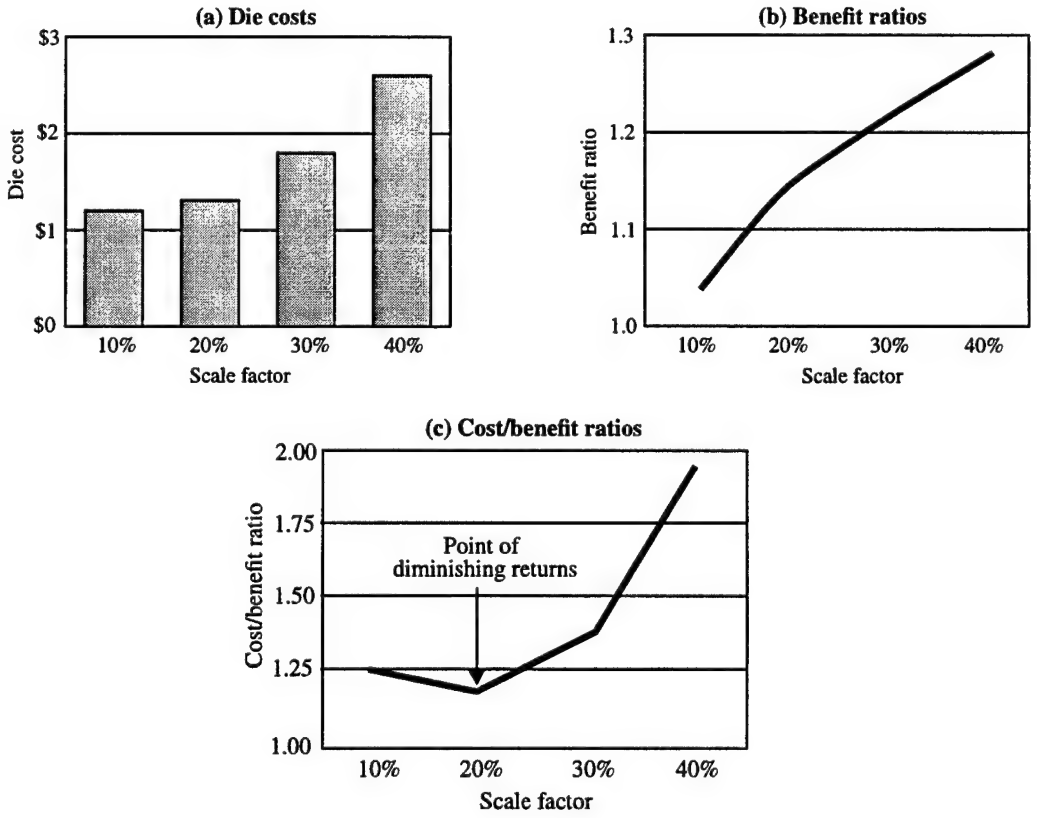


Fig. 2.9 Formation of a cost/benefit plot for a single design rule.

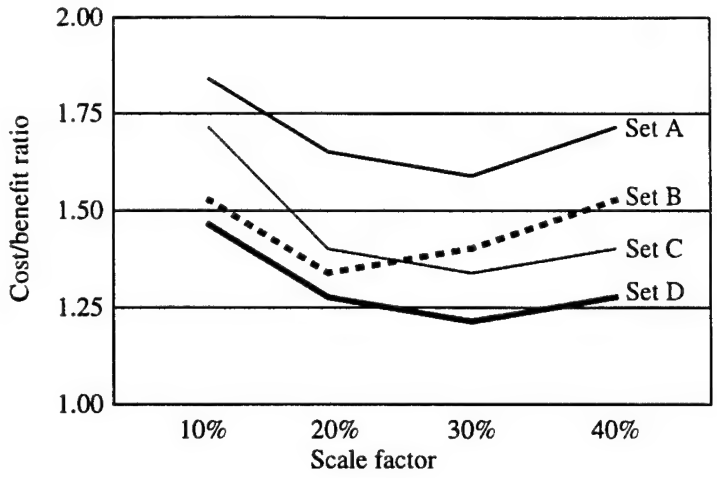


Fig. 2.10 Cost/benefit ratio plot of multiple design rules sets.

comparison of multiple design rule sets scaled through the same proportions. As indicated in the figure, reducing design rule set D by 30% provides the best cost-performance tradeoff among the four design rule sets under consideration. All other design rules should then be reconsidered for reduction once set D is reduced by 30%.

2.3.7 Defining and Comparing Multiple Scaling Iterations

A procedure for identifying and analyzing area-critical design rule sets has been established. By reducing the most cost efficient area-critical design rule set to its point of diminishing returns, a new baseline process is established for further analysis. The process scaling design space can be efficiently explored by iterating on this procedure, as illustrated in Table 2.3. During each iteration, only those design rule sets and scale factors that can be accomplished with the specified spending cap should be considered. At the conclusion of each iteration, a design rule set with a corresponding reduction ratio is created and the cost of scaling these design rules is subtracted from the spending cap. The iterations continue until the spending cap is reached, at which point a cost/benefit comparison can be made of all steps taken in the scaling process. This comparison determines the point of cost/benefit diminishing returns for the nonlinear scaling program as a whole.

Table 2.3 gives example of scaled design rule values produced at the completion of five iterations of the cost/benefit analysis procedure. For this example, a $100 spending

Design Rule	Normalized design rule values					
	Start	**Step 1**	**Step 2**	**Step 3**	**Step 4**	**Step 5**
Rule A	1.00	0.90	0.90	0.72	0.72	0.50
Rule B	1.00	1.00	0.90	0.90	0.81	0.81
Rule C	1.00	1.00	1.00	0.80	0.80	0.80
Rule D	1.00	0.90	0.81	0.65	0.65	0.46
Rule E	1.00	1.00	1.00	1.00	0.90	0.90
Rule F	1.00	1.00	1.00	0.80	0.80	0.56
Scale factor	-	10%	10%	20%	10%	30%
Scaling cost	-	$10	$15	$30	$15	$30
Spending cap	$100	$90	$75	$45	$30	$0
Cost/benefit ratio	-	5.0	4.5	**4.2**	5.1	6.0

Table 2.3 Multiple iterations of the costs/benefit analysis procedure.

cap is used. At the completion of the first iteration, design rules A and D are reduced by 10%, which costs $10 and therefore lowers the spending cap for the second iteration to $90. At the completion of the second iteration, design rules B and D are reduced by 10%, which costs $15 and reduces the spending cap to $75, and so forth. After five iterations, the spending cap is reached and the algorithm terminates. When comparing all five scaling steps taken during this analysis, the minimum cost/benefit ratio occurs after the third step. Therefore, the design rule scale factors that were generated by this step provide the most cost efficient improvement in performance and area.

This iterative procedure allows all design rules to be considered and reduced according to their ability to impact cost, performance, and area. Instead of performing a linear shrink of all design rules, this methodology produces a process where each design rule may be reduced by a different proportion. Table 2.4 illustrates this point by giving the overall reduction ratios at each step for each of the six design rules shown in Table 2.3.

Design Rule	Overall scale factor				
	Set 1	Set 2	Set 3	Set 4	Set 5
Rule A	10%	10%	28%	28%	50%
Rule B	0%	10%	10%	19%	19%
Rule C	0%	0%	20%	20%	20%
Rule D	10%	19%	35%	35%	54%
Rule E	0%	0%	0%	10%	10%
Rule F	0%	0%	20%	20%	46%

Table 2.4 Overall scale factors.

2.4 Conclusion

In the deep submicron era, manufacturing facility costs are growing exponentially. Under these conditions, managing scaling costs plays an increasingly significant role in achieving profitability. As global competition increases and profit margins decrease, it becomes crucial to maximize the revenue stream by achieving rapid time-to-market and making effective tradeoffs between cost and performance. Linear design rule scaling in the deep submicron era is not a sustainable exercise because of the prohibitively high costs associated with it. Nonlinear scaling allows a process to be scaled such that it provides more cost effective improvements in area and performance.

This chapter gives a methodology for selecting design rule reduction ratios that make an effective cost-performance-area tradeoff for a given technology. This technique, referred to as design rule cost/benefit analysis, is an iterative procedure that begins with identifying those design rules that have the greatest impact on layout area. Area-critical rule sets are then formed that are likely to provide the greatest improvement in area with minimal cost. The costs associated with reducing each design rule set is estimated, and the impact of reducing theses rules on overall performance and area is determined. Performance and area improvement data is combined with cost estimates to form a cost/ benefit ratio, which quantitatively describes the cost-performance-area advantage associated with design rule reduction. The slopes and inflection points of cost/benefit

plots can be used by the process engineer to select reduction ratios that make an effective tradeoff between cost and performance.

This procedure can be used to determine which design rules are the most area-critical, and how far they should be reduced before other design rules become more area-critical. Each iteration produces a baseline process for further analysis. By completing several iterations, each design rule is reduced according to its ability to impact cost, performance, and area. This technique concentrates on finding and scaling those design rules that provide significant improvements in performance and area with reasonable associated costs, while avoiding design rules that are too costly or do not provide significant performance and area improvements when scaled.

CHAPTER 3

PROCESS-INDEPENDENT RAM COMPILATION

A process-independent RAM compiler is an essential tool for conducting IC process design rule cost/benefit analysis. However, the capabilities of this tool must be enhanced to produce both near-optimal power-delay curves and optimized RAM layout. With these abilities, it is possible to make a fair and comprehensive comparison between different IC processes.

Process-independent RAM compilation includes the techniques and tools for generating and optimizing RAM layout for any IC process. A process-independent RAM compiler takes as input a description of the manufacturing process (including physical design rules), a representative device model, and RAM capacity, organization, and timing characteristics. The output is a DRC- and LVS-correct RAM layout that has been optimized for the user-specified IC manufacturing process. The primary difference between process-independent RAM compilation and a more traditional technique is that the former can specify both the process design rules and the device model at run-time, as illustrated in Fig. 3.1.

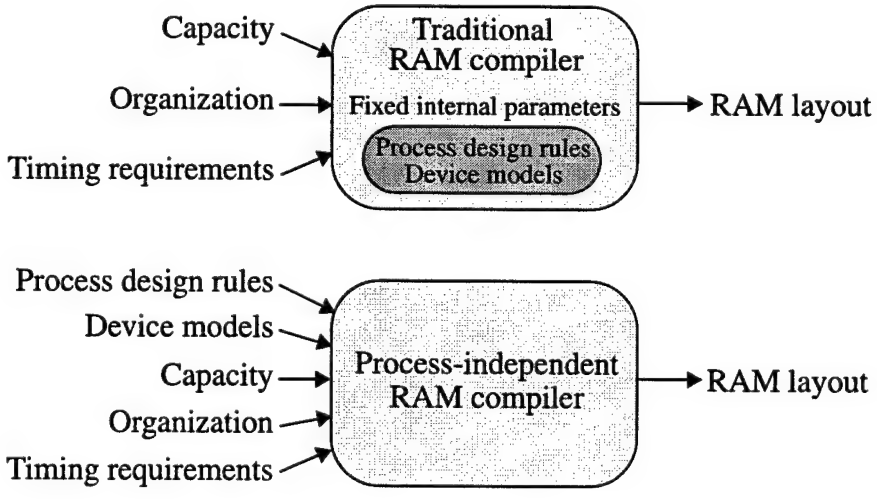


Fig. 3.1 RAM compilation techniques.

This chapter describes the organization, methodologies, and capabilities of the PUMA process-independent optimizing RAM compiler, a tool designed specifically for use in design rule cost/benefit analysis. Section 3.1 describes RAM compilation techniques and transistor sizing algorithms that have been employed in the past. This section also describes layout compaction methodologies and tools that are useful in generating optimized layout for any IC process. An overview of the structure, organization, and techniques used by the PUMA RAM compiler is given in section 3.2, while a description of the CGaAs circuit designs used by the RAM compiler are given in section 3.3. Section 3.4 describes a process-independent, pitch-matched RAM cell library generation technique, and section 3.5 gives the RAM compiler's transistor size optimization algorithm. Section 3.6 contains a description of a stability verification methodology utilized by the RAM compiler. Section 3.7 describes the RAM compiler's power rail sizing methodology and cell tiling, encoding, and routing capabilities. This section also gives some examples of RAM layout generated by the compiler. Finally, conclusions are made in section 3.8.

3.1 RAM Compilation and Layout Compaction

The regular and repeating structure of a RAM makes it an ideal candidate for automated layout generation and optimization. Such tools, known as RAM compilers, are capable of automatically creating an entire RAM layout that meets a set of user-defined capacity, organization, and performance objectives. Creating a RAM layout that meets these objectives in a specific manufacturing process requires four steps. First, transistor and power rail sizes must be appropriately chosen so that the RAM can meet the specified performance objectives. Second, a corresponding pitch-matched RAM cell library must be generated. Third, RAM leaf cells must be tiled into a configuration according to the user-defined capacity and organization specifications. Finally, row and column addresses must be encoded for correct functionality.

3.1.1 Transistor Sizing Algorithms

Once a RAM circuit design has been selected, the individual transistors must be sized to reach the targeted performance levels. Therefore, the heart of an effective RAM compiler is its transistor sizing engine. This tool determines a set of transistor sizes for

the RAM core and peripheral circuits that allow the RAM to meet the specified performance objectives. Since transistor sizes have a direct impact on speed, power, and layout area, an effective transistor sizing algorithm should be capable of optimizing a design based on a combination of these factors.

Many transistor sizing algorithms have been developed to optimize the power and/ or area of generic logic structures subject to timing constraints. These logic structures are given in the form of a circuit netlist annotated with parasitic capacitances and resistances. There are four general approaches to solving the generic logic transistor sizing problem. The first method transforms the circuit structure into a distributed RC network and models the delay through it as a classical nonlinear optimization problem. Geometric programming techniques are then used to find a mathematically exact solution. Arriving at this solution often involves defining the nonlinear optimization problem in terms of posynomial functions that can be transformed into a set of convex functions by means of an exponential transformation. A variety of exact or numerical methods can then be employed to find a local minimum, and hence, the global minimum of each convex subproblem. Several algorithms and sizing tools [73-79] have been developed that employ this technique. However, this method suffers from its inherent complexity and large computational requirements. Furthermore, it may not converge to a solution if the initial optimization problem is sufficiently complex. It should also be noted that these solutions are only exact for the parasitics used to construct the original distributed RC network.

The second technique employs heuristic methods to achieve near-optimal solutions to the nonlinear optimization problem [80-82]. This method identifies those transistors that are most sensitive to the objective, and incrementally sizes them in order to achieve the desired effects without violating the constraints. Heuristic algorithms cannot guarantee optimal results, but often come close, and require far less computation than rigorous mathematically exact methods.

The third approach combines "exact" and heuristic methods to achieve good results that have reasonable computational requirements. These hybrid techniques [83,84] use geometric programming methods to provide an initial starting point, then use heuristic methods to find a near-optimal solution. Thus, a more accurate solution can be obtained with only a small increase in computation time.

The fourth transistor sizing method is based on a stochastic modeling of circuit responses [85]. In this technique the values of the complex nonlinear objective functions are viewed as random variables with known distributions. This view accounts for variations in process parameters. Stochastic modeling allows the algorithm to globally view and efficiently optimize the entire circuit instead of just the speed-critical path. However, this approach resembles heuristic methods since it can only provide near-optimal solutions.

Several other interesting capabilities have been introduced into transistor sizing tools. Some of these capabilities include using SPICE to conduct iterative circuit simulations [86], using a static timing analyzer to identify speed-critical circuitry [87,88], using an interactive interface to allow the designer to control the direction of the search efforts [89-91], allowing multiple logic styles to be introduced into the design [92], providing gate-based instead of transistor-based optimization [93], and optimizing transistors in a multiple threshold voltage technology [94]. The most powerful sizing tools, however, extract parasitic capacitances and resistances from the layout and conduct both transistor sizing and layout compaction in their optimizing loops [95-98].

RAM core and peripheral transistor sizing can be viewed as a subset of the traditional transistor sizing problem. A RAM has a fixed organization that is known beforehand. Therefore, its sizing engine can employ specialized heuristics that are both accurate and fast. The sensitivities of various transistors on read and write access times are well understood, making the judicious choice of transistors for incremental adjustment comparatively easy. A RAM compiler combines RAM transistor sizing and layout generation capabilities into a single tool.

3.1.2 Past RAM Compilers

The first published RAM compiler, known as RAMGEN [99], was developed at Texas Instruments in 1986. This tool simply connected previously designed leaf cells into a parameterized RAM configuration that could be fabricated with either a 2.0 or 3.0 μm CMOS process. It also generated corresponding HDL descriptions and performance datasheets of the design. No iterative analysis or rebuilding steps were conducted to optimize the design because the tool was strictly a layout generator.

Since RAMGEN's introduction, numerous RAM compilers have been developed. In 1990, Texas Instruments completed a BiCMOS multi-port SRAM compiler [100,101] capable of generating designs with up to 8 k-bits of capacity. Although this tool was capable of producing a larger variety of SRAMs, it was still just a layout generator that did not employ iterative analysis or optimization steps.

In this same year, VLSI Technology, Inc. and LSI Logic Corp. introduced SRAM compilers for their ASIC customers. The VLSI Technology compiler [102] could build memories with up to 128 k-bits of capacity in a 1.0 μm CMOS technology. It used a proprietary silicon compilation language known as "SLICE" to both generate and connect SRAM leaf cells. This provided the customer with greater ease and flexibility in altering memory organizations. The LSI Logic MEMCOMP compiler [103] could build memories with up to six ports in either a 1.5 μm or 1.0 μm CMOS process. It used predesigned full-custom leaf cells as its building blocks. Both of these tools, however, were still just layout generators that did not employ iterative optimization techniques.

A breakthrough in RAM compiler technology was made in 1990 when Mitsubishi Electronics introduced their MAC2 memory compiler based on Cascade Design Automation's Compiler Development System (CDS) [104]. CDS is a silicon compilation language that provides process independence by parameterizing both device sizes and technology design rules. This compiler was also the first to use an iterative, constraint-driven optimization technique for buffer sizing. During each iteration, predefined lookup tables were used to help speed up the analysis of a given layout. This tool was capable of creating SRAM designs with up to six ports and 36 k-bits of capacity.

In 1991, Motorola's ASIC division, in cooperation with Mentor Graphics Corp., developed the Memorist SRAM compiler [105]. Memorist was based on Mentor's GDT tool suite and was capable of designing 0.7 μm CMOS SRAMs with up to two ports and 256 k-bits of capacity. It resembled the MAC2 compiler by implementing an iterative optimization technique to size transistors, but employed this technique in sizing the entire critical path circuitry instead of just the buffers. Memorist had another key advantage over MAC2 by allowing the designer to choose between using predefined lookup tables or SPICE to analyze critical path circuitry.

Another notable RAM compiler is Cascade Design Automation's HSLP™ (High Speed Low Power) CMOS SRAM compiler, which was developed as a part of their EPOCH silicon compiler. This compiler was similar to the MAC2 compiler in that it utilized CDS to generate, pitch-match, and tile leaf cells into an SRAM configuration. However, since HSLP was developed in-house, it employed additional, undocumented CDS features which were not available to the MAC2 developers. This tool also implemented an optimizing, iterative transistor sizing algorithm and device lookup tables to meet required timing constraints.

The first GaAs SRAM compiler, known as Aurora [106,107], was developed at the University of Michigan in 1994. This tool was capable of generating and optimizing E/D MESFET SRAMs having up to 8 k-bits of capacity. A novel current mirror memory cell [108] was employed as the storage node in these designs. The compiler resembled MAC2 by using CDS to both generate and connect SRAM leaf cells. It also combined the additional capabilities of the Memorist compiler by employing SPICE as the primary circuit analysis tool. Aurora implemented an iterative, constraint-driven transistor sizing algorithm to build memories that were optimized for speed, power, area or a combination of these metrics. Its heuristic gradient search algorithm also allowed a large power-delay-area design space to be searched rapidly.

3.1.3 Current Commercial RAM Compilers

There are two categories of commercial RAM compilers available today: gate-array compilers and "all-layer," or embedded compilers. Gate array RAM compilers have been developed by commercial ASIC vendors such as Texas Instruments and Motorola, and also by CAD companies such as Synopsys. These compilers have been designed to create memories from metal programmable gate array ICs. Synopsys' Cell-based-array Memory Architect™ provides RAM designs that can implement several configurations including single or multi-port, speed or power optimized, and synchronous or asynchronous timing.

Companies that are currently supplying embedded CMOS RAM compilers include Mentor Graphics, Cadence Design Systems, Artisan Components, Duet Technologies, Silicon Access, Micro Magic, Virage Logic, Virtual Silicon Technologies, Nurlogic Design, and Aspec Technology. Mentor Graphics' Memory Builder™ is a graphical

abstract floorplanner that can be used to create a parameterized memory generator. This tool iteratively assembles, extracts and characterizes memory layouts. It is not a true optimizing compiler, but merely a layout generator. However, it does provide the ability to create an optimizing front end that iteratively calls the tool to create variant RAM layouts.

Cadence's Structure Compiler™ is similar to the Mentor Graphics product. It is a tiling engine that can be used to generate layouts that are comprised of complex, repetitive structures that are either full-custom or parameterized leaf cells. This tool also has a programmable, procedural interface instead of providing its own optimizing front end.

One of the more advanced commercial RAM compilers is Artisan Components' Process Perfect™ memory generator. This "all-layer" compiler is capable of generating memory layouts that can be optimized according to a wide variety of design goals. It also implements an iterative, optimizing transistor sizing tool that uses a timing characterization curve-fitting technique that is within 2% of SPICE and a power characterization curve-fitting technique that is within 5% of SPICE. This compiler can also generate custom built-in self test (BIST) controllers for its embedded memories.

Most commercial RAM compilers, including those produced by Artisan Components, Silicon Access, Micro Magic, Virage Logic, Virtual Silicon Technologies, Nurlogic Design, and Aspec Technology are not truly process-independent [109]. These tools have been designed to produce optimized RAM layout for a specific process; they do not possess the ability to generate and optimize a RAM layout for any IC process that is specified at run-time. The exception to this is Duet Technology's IntelliMem™ compiler. This compiler was developed concurrently with the PUMA RAM compiler and employs a similar symbolic layout compaction technique to provide process independence.

3.1.4 Symbolic Layout Compaction

Full-custom IC design methodologies steer the circuit designer away from automated tools or techniques to produce circuit layout. The underlying assumption in full-custom design is that the human circuit designer will always be able to produce more compact layout than an automated layout generator. The major drawback in full-custom design is the amount of time and effort it requires. When a design is to be migrated from one process to another, much of this work must be repeated since the original mask designs correspond to the design rules of the original process.

Symbolic layout compaction is a technique for rapidly producing near-full-custom quality mask designs in a process-independent fashion. This approach involves using symbolic wells, transistors, contacts, vias, and wires to represent a circuit design rather than using DRC correct mask geometries. A compaction algorithm is then used to transform the symbolic layout representation into DRC correct mask geometries that correspond to design rule values in an external, user-modifiable file. The same circuit design can be quickly migrated to another process by changing the design rule values in the external file and re-executing the compaction algorithm. Layout migration, therefore, does not require altering the original symbolic layout representation.

Extensive research has been conducted in the area of symbolic layout compaction; an excellent summary of early work in this area is given by Boyer [110]. Numerous approaches were developed and studied during the early phases of these research efforts. Eventually, two main techniques emerged as the most viable solutions to the problem: constraint graph compaction and virtual grid compaction.

Constraint graph compaction [110-116] represents the corners, centerpoints, or edges of symbolic geometry using x- and y-coordinates. A directed constraint graph is then generated for the symbolic geometry that correctly represents the locality and connectivity relationships between the layout polygons. In the worst case, generating a directed constraint graph is bounded by $O(n^2)$ if every polygon is constrained by every other polygon in the layout. Since the complexity of the directed constraint graph impacts the efficiency of the layout compaction algorithm, various methodologies for creating minimized directed constraint graphs have been explored [111,114-116]. One such methodology, introduced by Lo and Varadarajan, is bounded by $O(n^{1.5}\log n)$ [115]. Once a minimized directed constraint graph is created, the area-critical path through the symbolic layout is determined and compressed. Most constraint graph compaction algorithms are one-dimensional, meaning they first compact the layout in the x- or y-dimension, then compact in the other dimension. The first two-dimensional compactor, which simultaneously compacts in the x- and y-dimensions, was introduced by Kim, Lee, and Shin in 1992 [113]. Finding and compressing the critical path in the layout can be performed iteratively to produce better results at the expense of additional computation time.

Virtual grid compaction [110,117,118] utilizes a more structured view of symbolic layout geometry. In this approach, the edges, corners, and centerpoints of symbolic geometry are placed on a virtual grid which is organized to establish the locality and connectivity requirements of the layout. For example, if two contacts are to have a fixed distance between them (in the y-direction), they will be placed on the virtual grid such that they share a common x-coordinate. During virtual grid compaction, all elements having the same x-coordinates are compared; those geometries that require the greatest amount of spacing will determine the minimum physical distance between those grid lines in the corresponding physical grid. This process is then repeated for all elements having the same y-coordinate. Virtual grid compaction is therefore a one-dimensional algorithm. Constraint graph compaction tends to produce more compact layout than virtual grid compaction, but virtual grid compaction is a less computationally demanding algorithm, and is especially well suited for the evaluation of hierarchical designs.

The ability to evaluate a hierarchical design while preserving pitch-matching requirements has greatly increased the ability of symbolic layout compactors to solve practical problems. Numerous approaches to hierarchical design compaction have been explored [114,118-129]. Two major techniques have emerged and are the most widely recognized. The first of these techniques is an extension, or modification of the constraint graph and/or virtual grid compaction algorithms that enables them to handle multiple levels of hierarchy. In the case of the constraint graph compaction method, a separate constraint graph is created for each leaf, intermediate, and top level cell in the hierarchy. The entire hierarchical structure is then evaluated bottom-up; once the lowest level cells are compacted, the next highest level cell is compacted, and so forth. Iteration through this process is likely to produce improved results at the expense of additional computation time. This hierarchical approach can also be used with the virtual grid compaction method [118].

The second approach to hierarchical design compaction involves viewing the hierarchical pitch-matching problem as an integer linear programming problem. Unfortunately, this problem, as it relates to layout pitch-matching, belongs to the class of problems that are NP-complete. Therefore, various approaches have been explored to reduce the amount of computation time required to produce good results. One of the most

efficient of these techniques breaks down the pitch-matching problem into several smaller problems by decomposing the layout into recurring structures and solving each structure in parallel [128].

Additional capabilities and improvements to existing layout compaction methodologies have been recently explored. One of these includes the ability to make changes to the topology, or shape of the layout block [130,131]. Another improvement involves compacting the layout based on timing critical paths instead of area critical paths [132]. In this technique, a timing analyzer is used to first determine the location of the delay-critical path in the layout, then the layout associated with this speed path is compacted. The process is repeated to produce a minimally sized layout with minimized delay.

3.1.5 Current Commercial Layout Compaction Tools

The proliferation of symbolic layout compaction research has led to the development of several commercial layout compaction tools. The most popular include Sagentec's DREAM™ (Design Rule Enforcement And Migration), RUBICAD's LACE™ (Layout Conversion Environment), and Duet Technology's MasterPort™ [35-39,134].

Sagentec's DREAM is capable of compacting standard cell libraries, memories, datapaths, PLAs, and large random blocks. Standard cells can be created for any IC process with a user-specified height and gridding characteristic. The user also has direct control over transistor, signal wire, and power rail widths. DREAM supports hierarchical compaction and pitch-matching for memories, datapaths, and PLAs. It is also capable of migrating large random blocks with up to 200,000 transistors by automatically partitioning the design into smaller blocks and compacting them in parallel. The compacted blocks are then reassembled once each block has been successfully compacted. Transistor sizing is accomplished by using a global scaling factor with an associated minimum and maximum size, referring to a user-defined scaling formula, or using lookup tables. Wire widths are also sized by using a global scaling factor or by referring to an annotated circuit netlist.

RUBICAD's LACE is very similar to DREAM, and has the ability to port standard cell libraries, hierarchical designs, and large random blocks. It also exploits the

parallelism inherent to compacting large blocks and allows the user to specify the device and wire sizes in the compacted design. RUBICAD describes LACE's performance as being bounded by O($n$log$n$).

MasterPort was originally developed at Cascade Design Automation (CDA) and was commercially available for several years. After CDA was purchased by Duet Technologies, Inc., MasterPort's commercial availability was limited; it is now being used primarily as an in-house cell library development tool. MasterPort utilizes an iterative hierarchical constraint graph compaction algorithm. Because of the iterative nature of this compaction algorithm, MasterPort is capable of producing good results, but at a high computational cost. The PUMA RAM compiler uses MasterPort to create RAM cell libraries in a process-independent fashion. Appendix A describes the manner in which MasterPort imports, constrains, and evaluates symbolic layout geometry to produce compacted, DRC-correct layout for any IC process.

3.1.6 Full-chip Layout Migration

The capabilities of symbolic layout compaction tools have steadily increased in their scope. Tools such as DREAM and LACE, as well as other research-oriented tools (such as DECOMP [136]) can successfully perform the compaction and migration of entire VLSI designs. This is accomplished by first compacting individual modules within the design, then re-placing and re-routing the modules according to the new process design rules. However, porting an entire VLSI design from one process to another involves much more than producing a compacted layout in the new process. Each transistor, signal wire, and power rail must be appropriately resized to ensure correct and optimal performance. DREAM and LACE make allowances for this by providing a means for the user to specify the new transistor sizes and wire widths through lookup tables or annotated netlists. However, these tools do not actually determine what these values should be.

To date, only one full-chip layout migration tool has been published that determines a set of optimized transistor sizes as a part of porting a design from one process to another. This tool, developed by Kishida et al. in 1994 [137], employs a two-phase optimization methodology. During the first phase, the VLSI layout is read and converted into a symbolic representation. The symbolic layout is then compacted

according to the target process design rules while maintaining the original transistor widths. The parasitic elements associated with the compacted layout are then extracted and a representative circuit netlist is generated. A timing analysis is then performed using the annotated netlist to determine a set of optimized transistor widths. This timing analysis involves reducing all transistor widths by a proportional scaling factor K, while ensuring that the scaled transistor widths do not fall below a user specified threshold, W_{th}. The simulated delay is then recorded and a cost function (W_T) is calculated for that particular design. W_T is the sum of all transistor widths in the migrated design and, according to the authors, is directly proportional to the sum of the design's static and dynamic power dissipation. This timing analysis is conducted for several different values of K in an effort to find the transistor width scaling proportion that allows the design to meet a specified timing objective with minimized power dissipation.

Once a set of optimized transistor widths has been determined, the optimization tool enters the second phase by re-compacting the symbolic layout and extracting the associated parasitic elements. An annotated circuit netlist is then generated and a timing analysis is conducted to ensure that the timing objective is met. The tool therefore provides an optimized, compacted layout in the target process.

This optimization tool suffers from several shortcomings. First, signal wire and power rail sizing is not performed during the optimization process. Second, a global proportional scaling factor is used to explore the transistor design space. This technique does not allow individual transistors, or sets of transistors, to be sized by different scale factors. Therefore, the portion of the design space that is explored by this tool is relatively small, limiting the ability of the tool to optimize results. Finally, the author's assumption that the design with the smallest total transistor width will be the design with the lowest power dissipation is a generalization that is not always correct. This is especially true in the case where a small gate is driving the input of a large one. Nevertheless, this optimization tool was the first to bridge full-chip layout migration and transistor size optimization.

3.2 The PUMA Process-independent RAM Compiler

The PUMA RAM Compiler was developed for two main purposes. First, to make fair and comprehensive comparisons of RAM performance and area in multiple processes for design rule cost/benefit analysis. Second, to supply optimized CGaAs embedded cache memories for the PUMA PowerPC microprocessor.

Producing an optimized RAM layout in a process-independent fashion is accomplished in two stages, as illustrated in Fig. 3.2. The goal of the first stage is to produce a near-optimal power-delay curve. This curve represents the achievable RAM power dissipation levels for a range of delay values in the specified IC process. Accurate power and delay data for use in cost/benefit analysis can be obtained from this curve, as described in Chapter 2. During the second stage, a near-optimal design description is created, which is a set of device sizes for the point on the power-delay curve that corresponds to the target cycle time. The RAM cell library that implements the near-optimal design description is then created, power rails are sized, and leaf cells are tiled into the configuration specified by the user. At this point, area data is available for use in cost/benefit analysis. By following this methodology, the RAM compiler is capable of producing a RAM layout for any process, with a specified capacity and configuration, while meeting a target cycle time with minimal power dissipation for the circuit design being utilized.

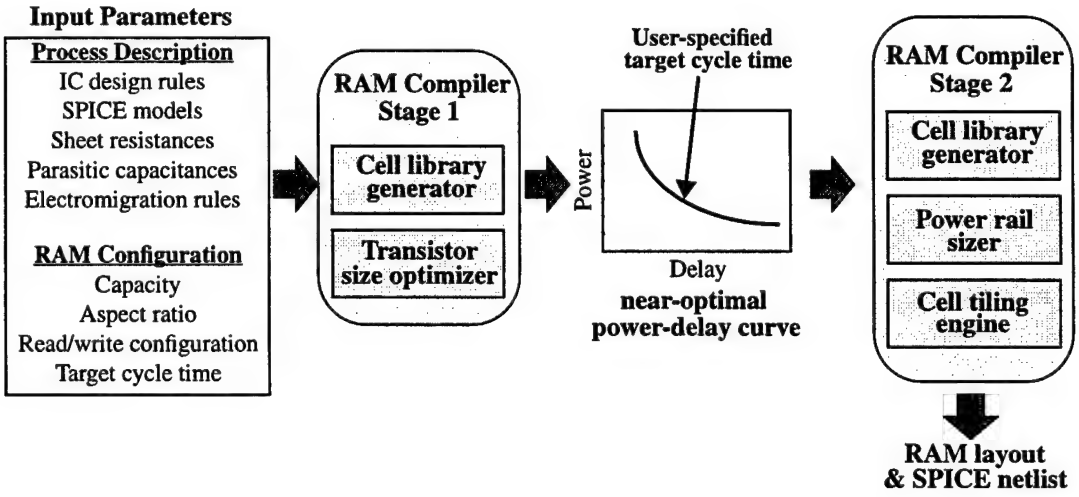


Fig. 3.2 The PUMA RAM compilation methodology.

3.2.1 Input Parameters

The input parameters to the PUMA RAM compiler include a description of the target IC process and the organization of the RAM. Process description parameters include the process design rule values, a representative SPICE model, sheet resistances for the interconnect layers, plate, perimeter, and fringe capacitance values for use in parasitic extraction, and maximum current density rules for electromigration. RAM configuration parameters include the number of rows and columns (which describe capacity and aspect ratio), the read/write configuration to be implemented, and the target cycle time. The PUMA RAM compiler is capable of supporting up to two different write conditions. The primary write condition can be configured to support up to a 4:1 column folding ratio while the secondary write condition can be configured to support up to a 16:1 column folding ratio. These write conditions allow the RAM to support a combination of block-, word-, and byte-write capabilities.

3.2.2 Major Components

As indicated in Fig. 3.2, there are four major components of the PUMA RAM compiler: the cell library generator, the transistor size optimizer, the power rail sizer, and the cell tiling engine. The cell library generator uses MasterPort to create a compacted, pitch-matched RAM cell library for the target process. The transistor size optimizer determines the sets of transistor sizes that correspond to various points along the near-optimal power-delay curve. The power rail sizer determines the minimum widths for the RAM power rails based on electromigration and maximum voltage drop requirements. Finally, the cell tiling engine creates the final RAM layout by instantiating cells from the cell library, placing vias to encode row and column addresses, and routing the row, column, and write signal buffers to the RAM core.

These four major components access a common library of 10 linked C functions through seven command-line executable programs. The C programming language is used to remain compatible with Duet's Compiler Development System^{TM} (CDS), which is a collection of C system calls that can access and instantiate MasterPort-generated layout. Table 3.1 summarizes the content of the PUMA RAM compiler function library. The seven command-line executable programs that access these functions and the arguments that may be passed to them are given in Appendix B.

Function	Purpose
cds_layout	Tile MasterPort generated RAM cells into a user defined RAM configuration
spice_include	Generate an annotated RAM cell library netlist for use in SPICE simulation
layout_extract	Extract parasitics elements from the RAM cell library
spice_generate	Generate a representative SPICE file for use in simulating an entire RAM
cell_analyze	Determine the stability of a six transistor static RAM cell in the presence of noise on the power rails
optimize_buffer	Create a near-optimal power-delay curve for a row address, column address, or write signal buffer
optimize_row	Determine the transistor sizes for a pseudo-NMOS buffer stage that ensures an acceptably low V_{OLmax}
optimize_sense	Create a near-optimal power-gain curve for a differential voltage amplifier
optimize_ram	Create a near-optimal power-delay curve for an entire RAM
railsize	Determine the minimum acceptable widths for a RAM's power rails

Table 3.1 PUMA RAM compiler library functions.

3.2.3 RAM Organization

RAMs created by the PUMA RAM compiler are organized according to Fig. 3.3. The memory core is surrounded on three sides by peripheral circuitry. Column decoders, write buffers, and bit line prechargers are placed on the top side of the memory core. Row decoders and word line buffers are placed on the left side of the core and the sense amplifiers are placed on the bottom of the core. In this configuration, write data enters the RAM from the top and read data exits from the bottom.

3.3 CGaAs SRAM Circuit Design

There are two types of volatile random access memory (RAM): static and dynamic. Static random-access memory (SRAM) differs from its dynamic counterpart by not requiring a periodic refresh in order to keep a stored binary value. Since an SRAM cell stores its value in a pair of cross-coupled inverters, it has greater drive strength, at the

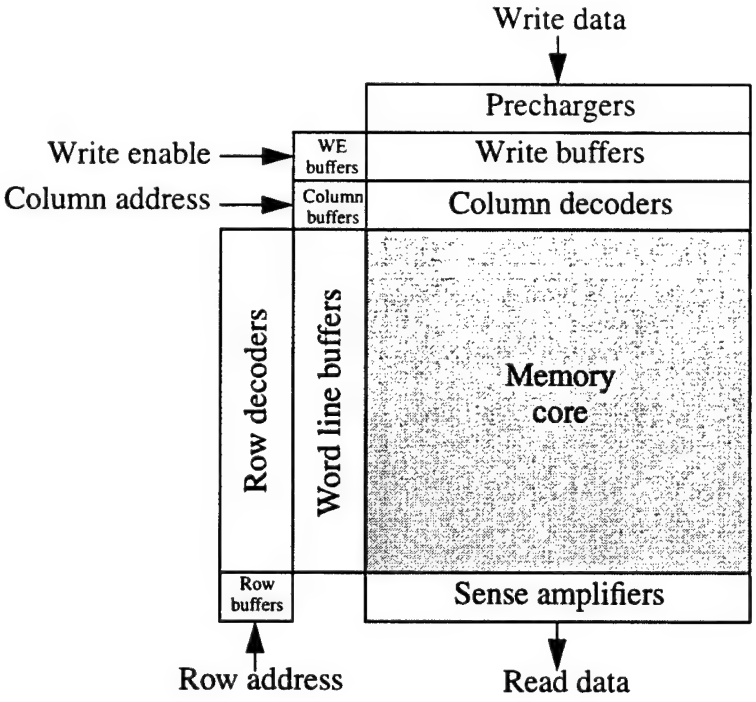


Fig. 3.3 RAM block diagram

expense of larger size, than a dynamic cell which stores its value on a capacitor. This leads to the implementation of memories that have faster access times than dynamic RAMs (DRAMs). SRAM can be easily embedded with a processor core because it does not require specialized processing steps to operate efficiently, as does one-transistor DRAM. The circuits required to restore destructive reads are also not required with SRAM since its read process is non-destructive. For these reasons SRAM has become the implementation technique of choice for embedded cache memory.

A typical six-transistor (6T) GaAs E/D MESFET SRAM cell utilizes cross-coupled inverters with depletion loads to store a binary value. Regardless of the data value being stored, there is always a conduction path from the power supply to ground in a cell with depletion loads. Consequently, this type of cell dissipates significant static power. A 6T CGaAs SRAM cell, on the other hand, utilizes cross coupled complementary inverters; its static power dissipation is due only to device leakage.

3.3.1 GaAs SRAM

Since their introduction in 1980 [138], GaAs-based SRAMs have been designed in many processing technologies, including C-JFET [139,140], E/D MESFET [141], and

CGaAs [142]. GaAs SRAMs are not yet competitive with silicon designs from density, capacity or efficiency standpoints. Much of this is due to the immaturity of current GaAs IC processing technologies which have courser design rule geometries and inferior integration levels.

The largest GaAs SRAM fabricated to date is a 64 k-bit monolithic chip designed for the Fujitsu VPP500 vector parallel processor [141] in 1991. This E/D MESFET DCFL design is very fast for its capacity, but dissipates 5.9 W. The most power efficient GaAs SRAM was designed in 1994 using Motorola's 0.7 μm CGaAs process. This 4 k-bit SRAM dissipates only 16.2 mW while providing a 5.3 ns access time [142].

The smallest GaAs 6T cell reported to date measures 236 μm^2 and was used in the previously mentioned Fujitsu 64 k-bit SRAM. This cell is over 30 times larger than the smallest reported silicon 6T SRAM cell. Despite this large difference, continual improvements in GaAs 6T SRAM cell area have been made, as illustrated in Fig. 3.4.

3.3.2 SRAM Circuit Design Issues

When deciding upon an SRAM circuit style or design technique, one must first consider the ramifications of the technology being used. From an understanding of the IC technology comes an estimate of the available integration levels and operating voltages. A RAM circuit designer must choose appropriate buffer styles for the technology being used (push-pull, full complementary, superbuffer, etc.) and develop a technique for optimally

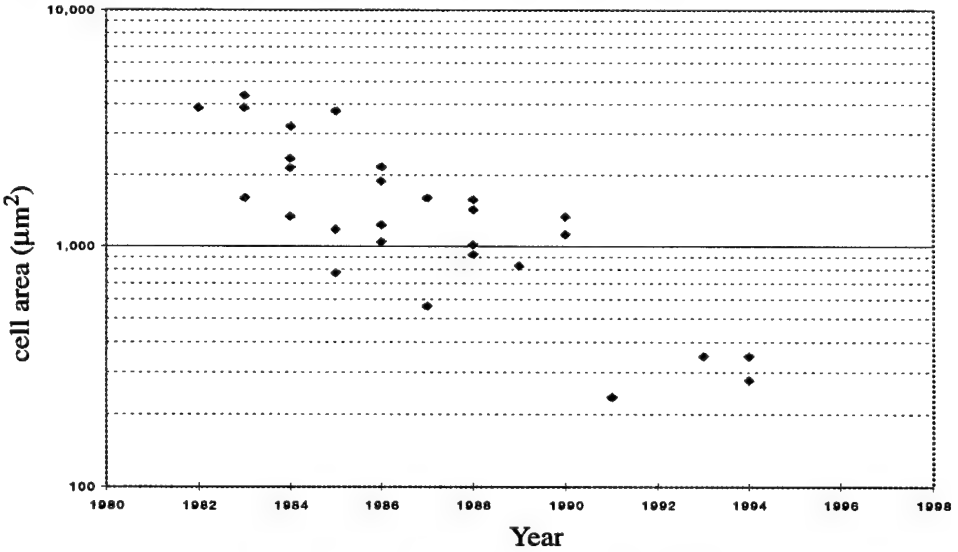


Fig. 3.4 GaAs 6T SRAM cell area

sizing them. The decision to use static or dynamic address decoding schemes has a large impact on overall circuit speed and complexity. Other circuit issues include precharging schemes (full V_{DD} or V_{DD} - V_{tn}), bit-line sensing schemes [143-146] (voltage-mode, current-mode or latching), multiplexer styles (restoring or pass-gate), and memory cell styles (static or dynamic, 4T or 6T, etc.). The choice of an appropriate operating mode [147] (asynchronous, synchronous, or wave-pipelined) is a crucial decision. RAM designers must also consider whether and how to use row and/or column redundancy [148] to increase yield.

3.3.3 CGaAs SRAM Circuit Design Objectives

Before the previously mentioned circuit design issues can be addressed, the primary and supporting design objectives must be appropriately defined for the target application. The PUMA CGaAs RAM compiler has two target applications: providing data for design rule cost/benefit analysis and providing optimized RAMs for the PUMA microprocessor. To meet the requirements for design rule cost benefit analysis, a circuit design must be chosen that is simple and robust; it must be a design that can be ported to a wide range of processes and still work correctly and reliably. Preliminary studies conducted on the PUMA microprocessor architecture suggested that the primary, off-chip data caches would be in the delay-critical path, and would therefore directly affect the maximum achievable operating frequency. In order to increase the maximum processor operating frequency, RAM circuits should also be designed for high-speed operation. These considerations, i.e. robustness and portability with fast access times, are the primary design objectives. Supporting objectives for the PUMA RAM compiler circuits include the need to minimize circuit area and complexity (since CGaAs is a low integration level technology) and to minimize power dissipation.

3.3.4 CGaAs 6T Memory Bit Cell

A 6T memory cell is used by the PUMA CGaAs RAM compiler because of its inherent stability and reliability. It is not feasible to implement a 4T memory bit cell with CGaAs because a highly resistive load layer is not available. Fig. 3.5 illustrates the layout of the CGaAs 6T SRAM cell. The ability to directly abut N- and P-diffusions in CGaAs

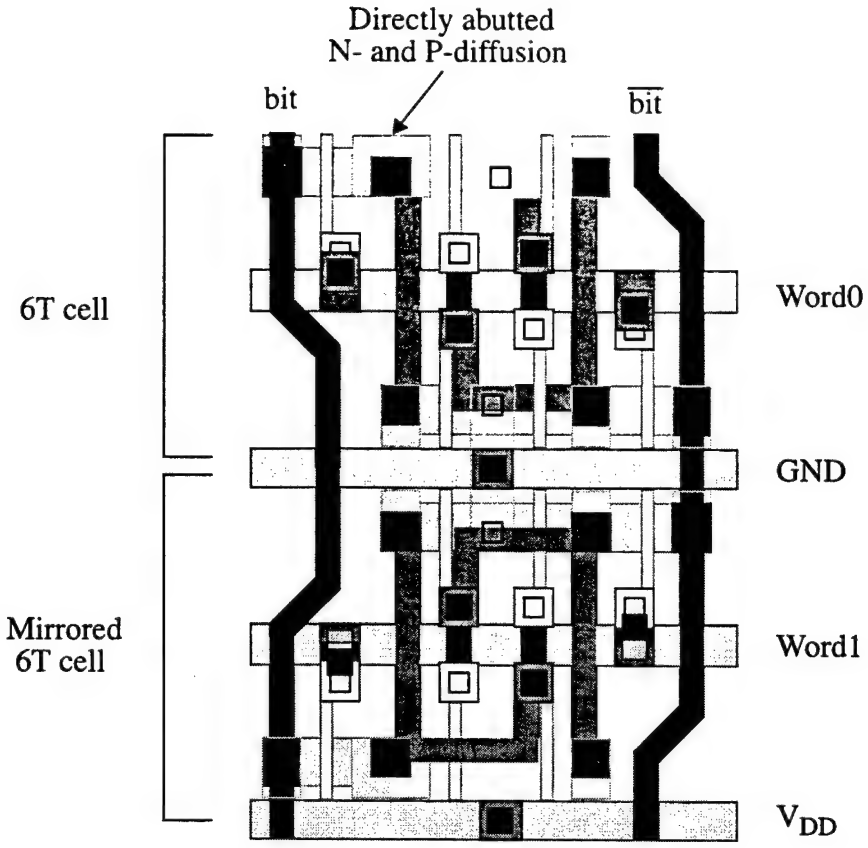


Fig. 3.5 Two CGaAs 6T memory bit cells.

provides a significant improvement in layout area, while the inability to connect an N- and P-diffusion with a single piece of gate metal increases the layout area.

There are several important and unique characteristics of the CGaAs 6T cell. First, the bit lines are located on the outside edges of the memory cell instead of in the center of the cell. There are advantages and disadvantages to this. A major advantage is the decrease in coupling capacitance between the bit and $\overline{bit}$ wires because these signals have been moved away from each other. This provides a boost in performance because bit and $\overline{bit}$ are complements of each other, which creates a miller capacitance effect between them. With the bit lines placed on the outside edges of the cell, however, the coupling and cross-talk capacitance between adjacent 6T cells increases. To counter this, the bit lines of the CGaAs 6T cell are kinked, which increases the average distance between bit lines of adjacent 6T cells.

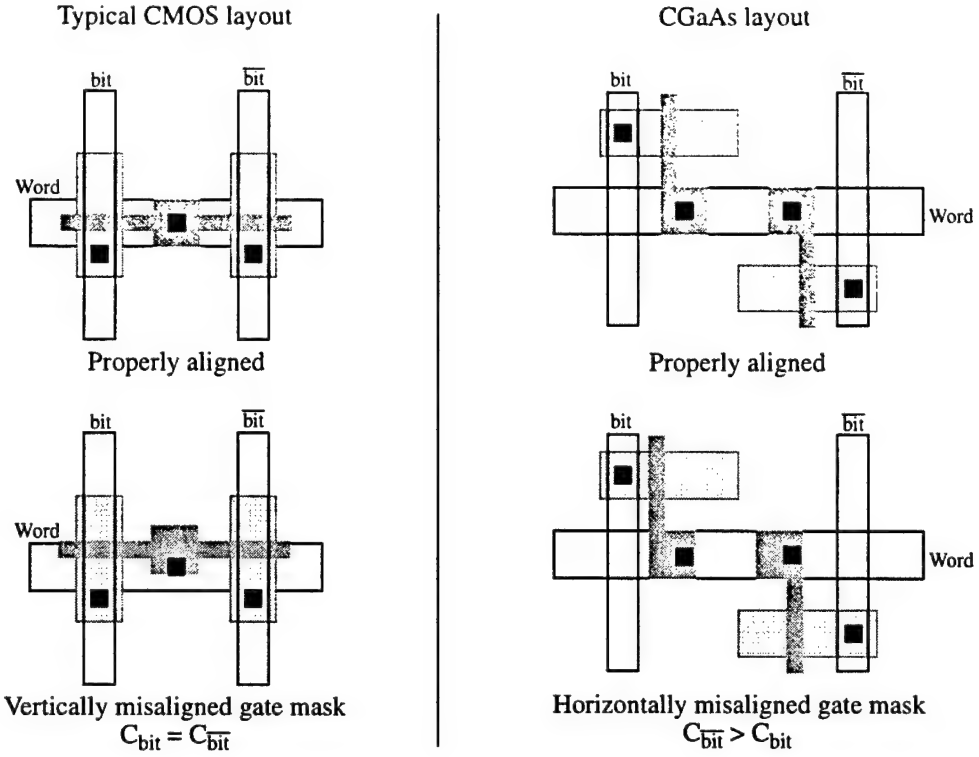


Fig. 3.6 6T access transistor and bit line layout.

In a typical 6T SRAM cell layout, the pass transistor diffusions and gates are similarly oriented, as illustrated in Fig. 3.6. This allows the capacitance associated with the two access transistor source diffusions, and their contribution to the total capacitance on the bit and $\overline{bit}$ wires, to track each other in the case of a vertical misalignment of the gate mask. In the CGaAs transistor 6T cell layout, the two access transistors are oppositely oriented. In the case of a horizontal misalignment of the gate mask, the total capacitance on one bit line will increase while the total capacitance on its complement will similarly decrease. As the capacitance of the pass transistor source decreases for bit, but increases for $\overline{bit}$, or visa-versa, an offset voltage will appear at the input to the sense amplifiers, resulting in an access time push-out.

To counter this affect, a bit-line crossing technique [149] is utilized to equalize the capacitance on the bit lines in the presence of a gate mask misalignment. This technique involves switching the polarity of the bit and $\overline{bit}$ lines in the memory array such that for half of the 6T cells in a column, bit is on the right and $\overline{bit}$ is on the left, and for the other half of the cells in the column, bit is on the left and $\overline{bit}$ is on the right, as shown in Fig. 3.7.

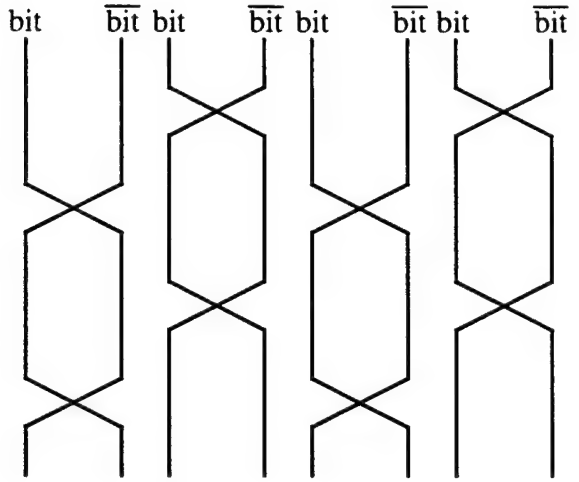


Fig. 3.7 Crossed bit line architecture.

There is another major advantage to crossing the bit lines that can be observed from Fig. 3.7. If the bit lines are crossed, a particular bit line in a column of 6T cells is equally coupled to bit and $\overline{bit}$ of both the left and right adjacent columns of 6T cells. Since each bit line is equally coupled to both bit and $\overline{bit}$ of the adjacent columns, the associated Miller capacitance between them is effectively eliminated.

3.3.5 CGaAs RAM Column Circuitry

As previously mentioned, the primary RAM design objectives are to create a robust, portable design with a fast access time. Synchronous RAMs have a distinct advantage over their asynchronous and wave-pipelined counterparts in their ability to meet these two primary design objectives. Unlike asynchronous designs, synchronous designs do not depend on self-timed circuits, the performance of which varies with the characteristics of the manufacturing process. Synchronous RAMs are therefore more simple than asynchronous and wave-pipelined designs. Because of their inherent simplicity, synchronous RAMs can also be very fast. The main disadvantage of using synchronous circuits is the larger power and area requirement associated with distributing clock signals throughout the design.

Fig. 3.8 illustrates the column circuitry used by the PUMA RAM compiler. The column circuitry is comprised of 6T memory cells, a sense amplifier, a write buffer, and a bit line precharger. During the first phase of the clock cycle, the precharger pullup transistors pull the bit lines toward V_{DD}. The equalizing PFET in the precharger is used to

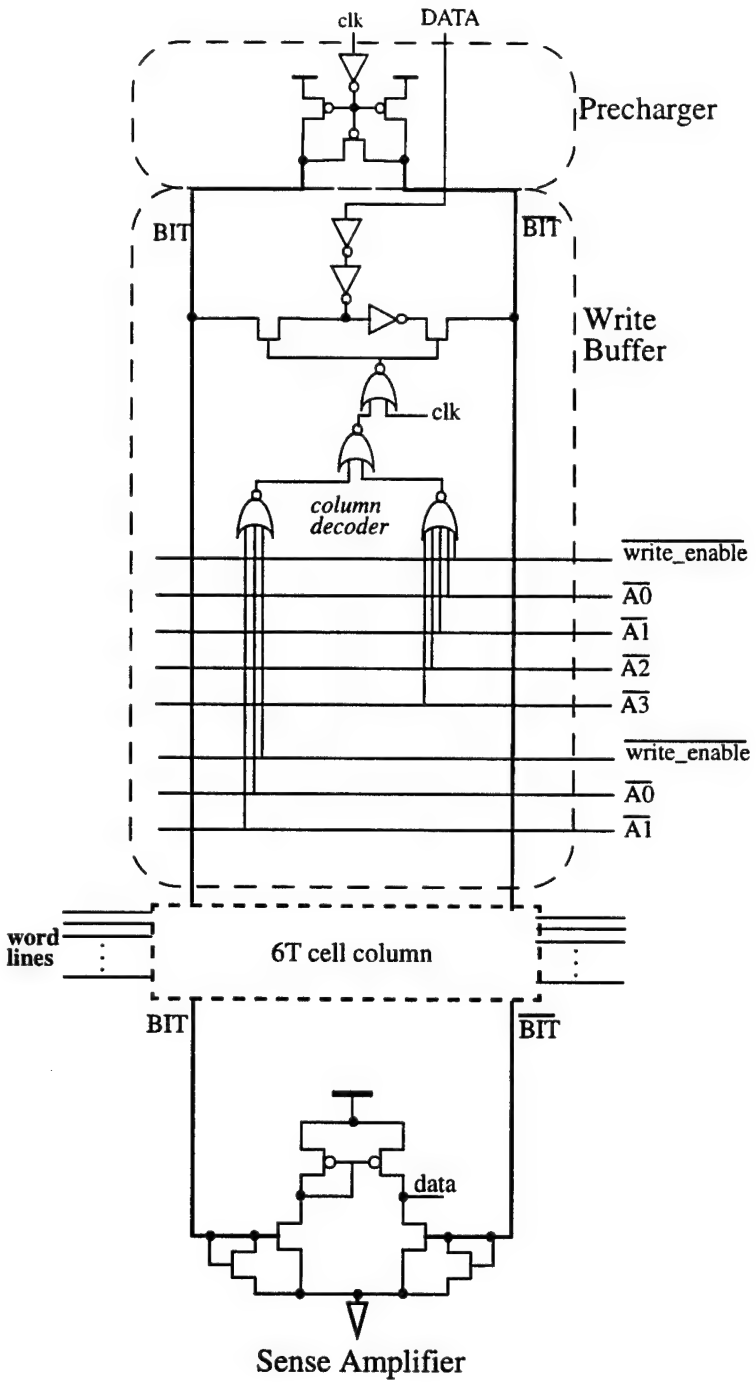


Fig. 3.8 CGaAs RAM column circuitry.

speed up the precharging process through charge sharing and to ensure that bit and $\overline{\text{bit}}$ begin the second phase of the clock cycle at equal voltage levels. An inverter is used to drive the precharge and equalization transistors to reduce the loading on the clock line.

During the second half of the clock cycle, a word line is asserted and either a read or write is performed. During a write cycle, data is driven onto the bit lines through large pass transistors that are enabled during the second half of the clock cycle if one of the two write conditions is satisfied. As indicated in the figure, one write condition can support up to a 16:1 column folding ratio, while the other can support up to a 4:1 column folding ratio. Pseudo-NMOS NOR-based decoding circuitry is used in the write buffer to eliminate the requirement for multiple FETs in series.

During a read operation, data from the asserted 6T memory cell is driven onto the bit lines and sensed by the differential voltage amplifier. The diode-connected NFETs are used to adjust the precharge point. By increasing the widths of these transistors, one can lower the precharge point of the bit lines and increase the gain of the sense amplifiers.

3.3.6 CGaAs Row Decoding and Word Line Buffering

During the first phase of the clock cycle, the row and column address lines are driven to the row and column decoders in preparation for driving the associated word lines and write buffers. Fig. 3.9 illustrates the row address decoding and word line buffer circuitry used by the PUMA CGaAs RAM compiler.

A static pseudo-NMOS NOR gate is used to decode the row address because of its simplicity. It also provides a fast decoding time because the NOR gate does not use FETs in series. The fan-in of the NOR-based decoder can be adjusted in increments of two by adding a pair of pulldown NFETs. The word line is driven high by a large PFET that is placed at the end of a series of inverters. The sizes of these inverters and the final driving PFET can be sized according to the number of columns in the RAM array.

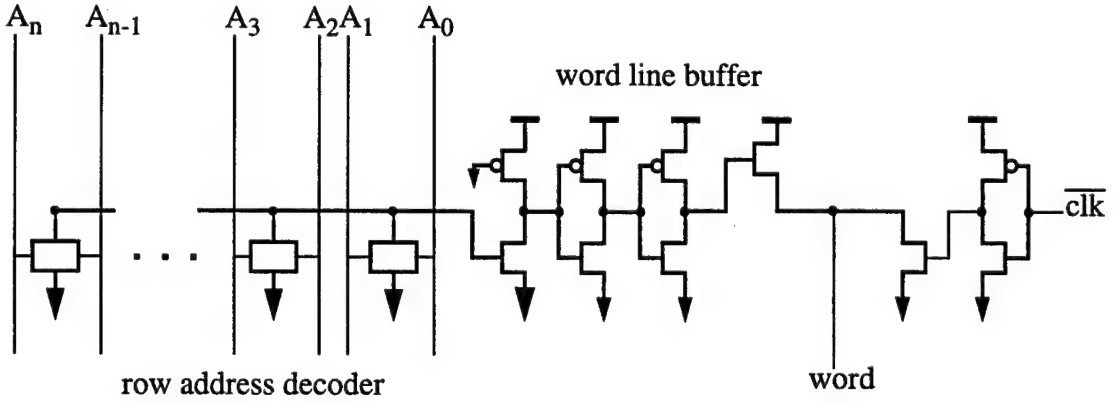


Fig. 3.9 CGaAs row decoding and word line buffer circuitry.

None of the word lines should be asserted during the first phase of the clock cycle when the bit lines are being precharged. Therefore, an appropriately sized NFET is used to pull down all the word lines toward ground during the first phase of the clock cycle. The pulldown transistor is driven by an inverter to reduce the loading on the clock line. This technique for synchronizing the word line pulse may seem like a high-power solution, but only a single word line is asserted at any given time, resulting in only one active crossbar circuit (where the pullup and pulldown transistors are both active). Furthermore, this crossbar circuit is only active shortly before and immediately following the transition to the second phase of the clock cycle.

3.3.7 CGaAs Address and Write Enable Buffers

The row and column address lines, and the write enable wires, are driven by the superbuffer circuitry illustrated in Fig. 3.10. These buffers were designed to provide low propagation delays for both large and small loads. When compared to a conventional multi-stage complementary buffer, the address and write signal superbuffer provides smaller propagation delays at the expense of higher power dissipation. The pullup NFET is used to boost the signal to V_{DD}-V_{tn}, and the pullup PFET pulls the signal to V_{DD}.

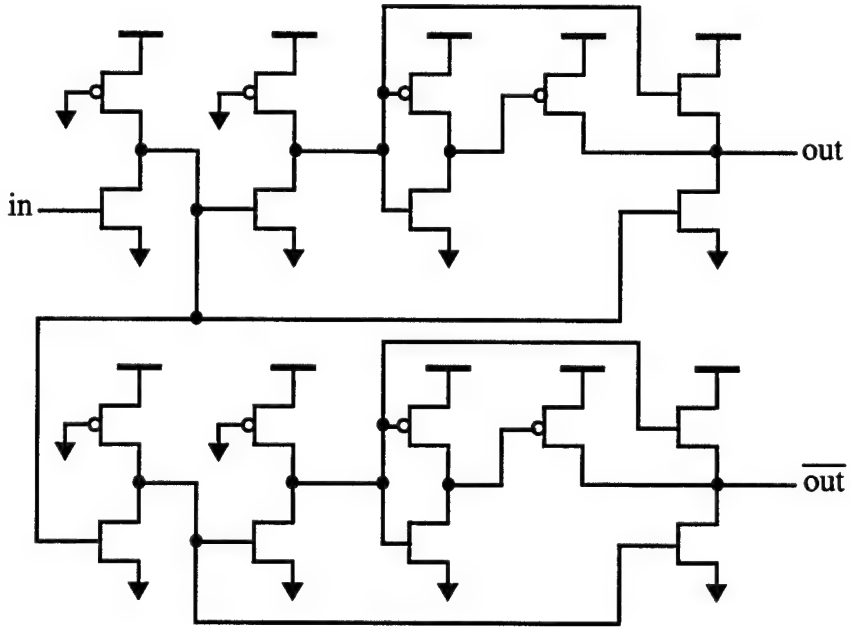


Fig. 3.10 Address and write enable superbuffer circuitry.

3.3.8 Circuit Simulation

Fig. 3.11 illustrates the simulation of a read operation for a synchronous RAM utilizing the previously described circuits. As indicated in the figure, the bit lines are precharged to about 1.0 V during the first phase of the clock cycle. During the second phase of the clock cycle, a word line is asserted and the bit lines move apart according to the data value stored in the corresponding 6T cell. As the voltage difference between the bit lines increases, the output of the sense amplifier is driven to indicate a binary "1".

The correct functionality of the PUMA RAM compiler circuits can be verified through SPICE-based timing simulation. These simulations can also be used to identify critical paths through the circuitry, which is useful in circuit optimization. To verify correct functionality, it is not adequate to simulate a single read and write operation. Instead, combinations of read and write operations that verify correct functionality over a range of sequential worst-case events need to be simulated [107].

There are four combinations of read/write activities to the same column that must be verified, including read-after-read (RAR), read-after-write (RAW), write-after-read (WAR), and write-after-write (WAW). Moreover, the data values being written or sensed should be of opposite polarity when considering back-to-back operations. This scenario introduces the worst-case situation where the voltage levels on the bit lines must undergo the greatest change in the shortest amount of time. Fig. 3.12 illustrates the representative structure and an associated timing diagram that can be used to verify correct RAM functionality across a range of worst-case sequential scenarios.

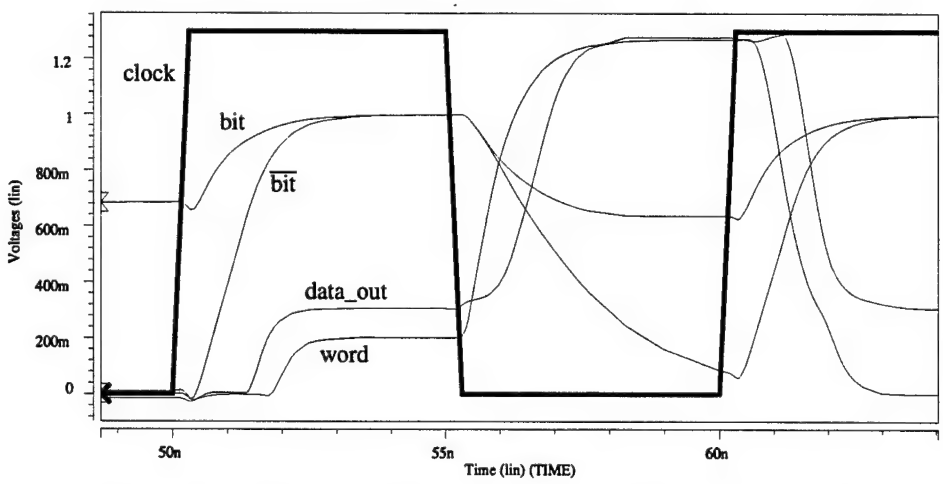


Fig. 3.11 CGaAs synchronous RAM cycle SPICE simulation.

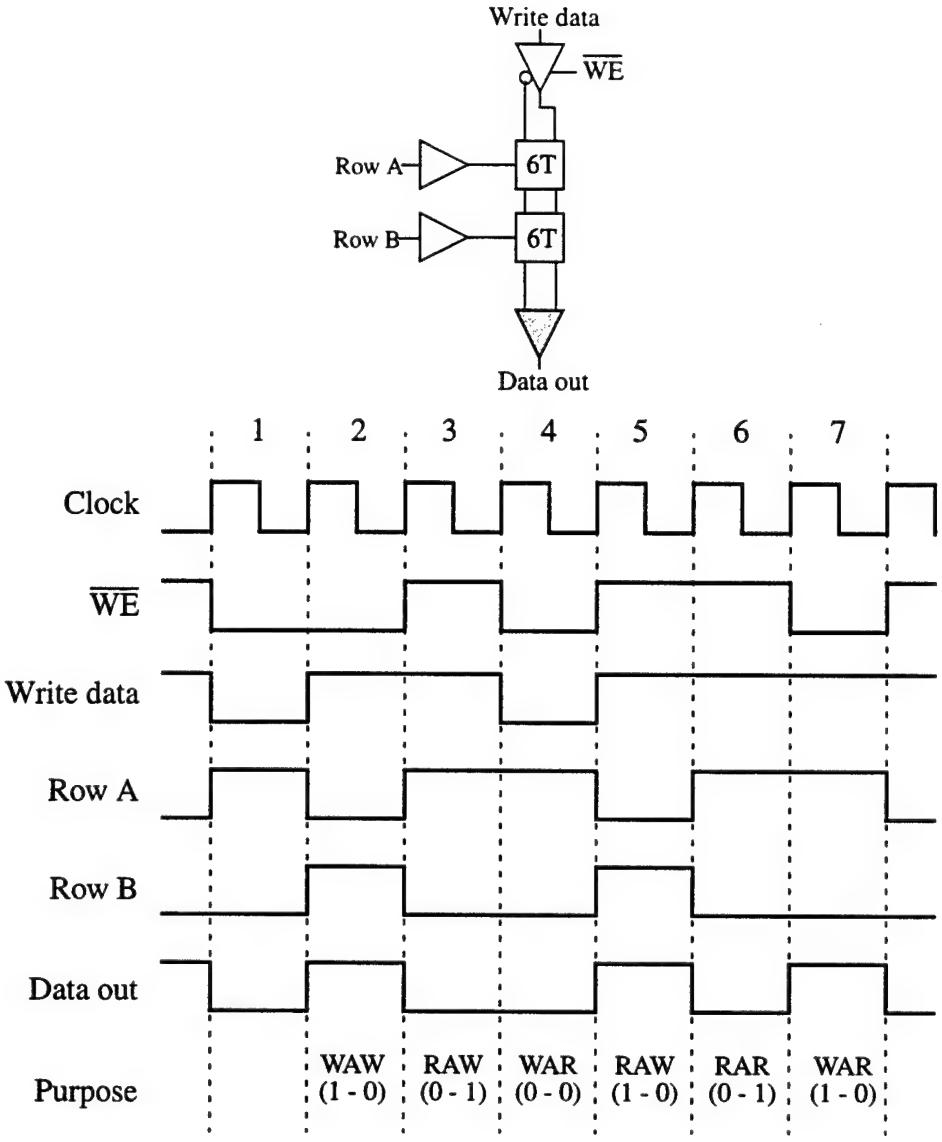


Fig. 3.12 RAM circuit simulation timing diagram.

The use of two cells on the same column allows opposing data values to be used in RAW, WAR, WAW, and RAR scenarios. RAW (0 - 1), for example, represents a read (data out = 0) following a write(1). For this simulation, cells A and B are set to initial values of 1 and 0 respectively. By comparing the observed output of the sense amplifier with the expected output for each of the seven cycles, it can be determined whether the RAM is capable of operating correctly at the simulated frequency. The circuit netlist, of course, needs to be correctly annotated with extracted capacitances and resistances for the results to accurately represent the operation of an entire RAM.

3.4 RAM Cell Library Generation

There are significant challenges associated with automatically generating RAM cell library layout for a range of IC processes. RAM leaf cells must pitch-match in both the vertical and horizontal dimensions in order to properly align. For example, a memory bit cell must align with column circuitry, including the sense amplifier and write buffer on one side, and with row circuitry, including word line buffers and row decoders on the adjacent side. Not only must RAM cells pitch-match, but their signal port locations must align to achieve correct connectivity. Generating an optimized and DRC/LVS correct RAM cell library for a range of IC processes can be accomplished with a hierarchical, layout compactor. The PUMA CGaAs RAM compiler utilizes MasterPort [150] to create compacted, pitch-matched RAM cell libraries.

3.4.1 MasterPort

MasterPort takes generic layout as its input, and creates compacted, DRC-correct layout. MasterPort automatically places constraints on imported layout geometries to control the manner in which layout compaction is accomplished. These constraints reference design rule micros, symbolic representations of minimum dimension and spacing requirements, which have values specified in a user-defined IC process description file. The key advantage of using MasterPort is that a single set of geometry constraints is valid over a wide range of design rule values. MasterPort also allows the designer to edit these constraints to solve topological issues that may arise when migrating between processes.

Imported and constrained cell geometry can be evaluated to produce compacted layout for different processes. The MasterPort technology database stores all relevant design rules for a given process. However, transistor size, power rail width and signal wire pitch values can be retrieved from a local text file prepared by the RAM compiler prior to cell evaluation. This gives the compiler complete latitude (within the constraints of the technology) to create a RAM cell library that is DRC/LVS-correct, and meets a particular power-delay-area requirement [151].

3.4.2 Hierarchical MasterPort

MasterPort, as well as other commercial hierarchical layout compactors including DREAM and LACE, have the ability to import, constrain, and evaluate a two-level hierarchical cell, which provides a way of generating a pitch-matched RAM cell library. To accomplish this, a hierarchical cell is imported and the individual leaf cells are constrained in the usual manner. Next, top-level cell constraints are placed on the leaf cell abutment boxes and port locations to establish pitch-matching and alignment requirements. Hierarchical evaluation is an iterative procedure that is accomplished by first evaluating the individual leaf cells and then determining whether the top-level constraints are met. If the constraints are not met, additional constraints are placed on the leaf cells to support the top-level constraints, and the evaluation process is repeated until all top-level constraints have been satisfied. Appendix A describes MasterPort's hierarchical evaluation process in greater detail.

3.4.3 RAM Supercell Organization

A pitch-matched RAM cell library can be created for a particular technology by importing, constraining and evaluating a RAM supercell. A supercell is a two-level hierarchical cell that is comprised of one or more instantiations of each unique leaf cell. For a RAM, these leaf cells include memory bit cells, sense amplifiers, buffers, decoders, power rails, and all other necessary RAM components. A supercell is organized to establish the pitch-matching requirements of each leaf cell. Fig. 3.13 illustrates the content and organization of the RAM supercell used by the PUMA RAM compiler. The supercell uses 54 device and interconnect size variables; 33 pertain to device sizes and 21 pertain to power rail widths and signal wire pitch. A RAM supercell need be imported and constrained only once. Thereafter, a new RAM cell library can be created by simply evaluating the supercell using the design rules for the target IC process.

3.4.4 Buffer Supercell Organization

The address and write signal buffer library is created by importing, constraining and evaluating a buffer supercell. Fig. 3.14 illustrates the content and organization of a buffer supercell. This supercell requires 26 user-defined device and interconnect variables, 23 of which determine device sizes while the remaining 3 determine power rail

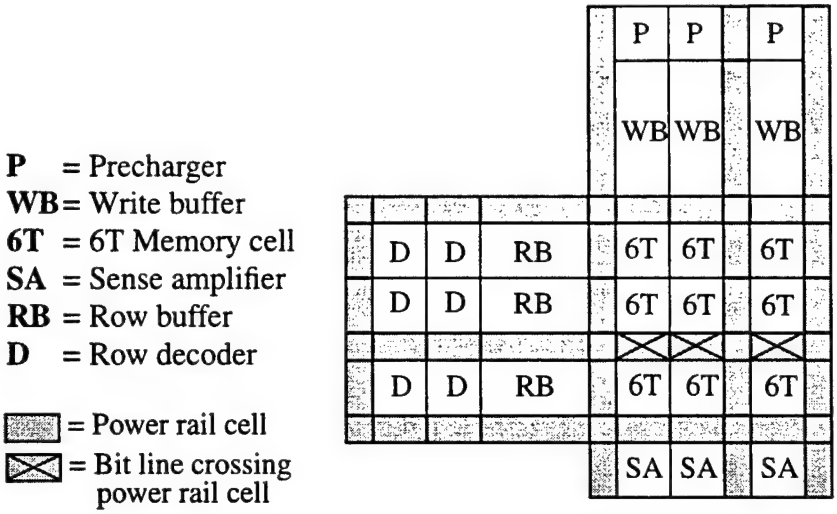


Fig. 3.13 RAM supercell content and organization.

widths. Though the row and column address buffers have the same structure, the transistors in each buffer are sized differently because the loading associated with the row and column address lines may be different. Therefore, the buffer supercell instantiates two versions of the same buffer, one for the row address lines and one for the column address lines. The write enable buffer is identical to the non-inverting address buffer. Fig. 3.15 is a plot of an evaluated RAM and buffer supercell for a 0.5 μm CGaAs process.

3.5 RAM Cell Library Optimization

The sizing of the transistors in a large circuit can be viewed as an optimization problem where an objective function is to be minimized (or maximized) subject to certain constraints. In the case of a RAM that is generated by the PUMA RAM compiler, the optimization problem is to minimize power dissipation subject to timing and reliability constraints. One method for solving this type of optimization problem is to globally view

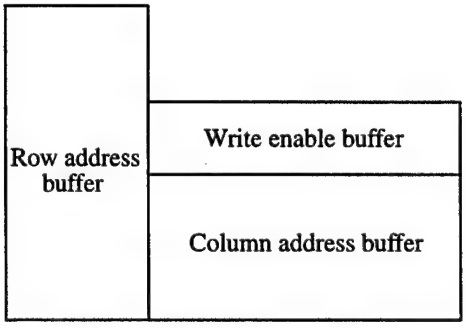


Fig. 3.14 Buffer supercell organization.

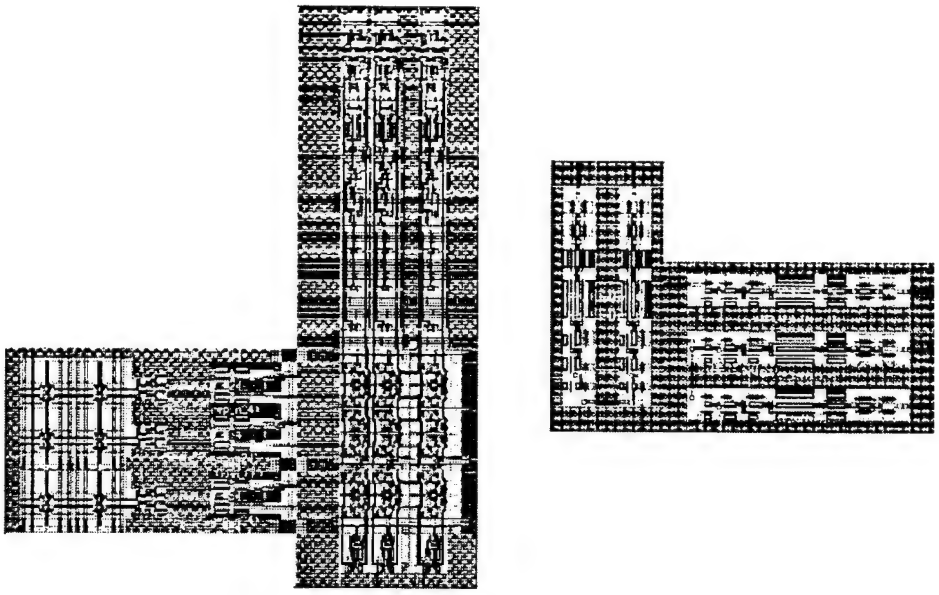


Fig. 3.15 Evaluated 0.5 μm CGaAs RAM (left) and buffer (right) supercells.

and optimize the entire circuit. This approach can be extremely complex and computationally expensive, especially if the circuit is large. Transistor sizing tools, such as HSPICE™, are capable of solving this type of transistor sizing problem as long as the timing and reliability constraints can be adequately described. HSPICE becomes unusable, however, if there are multiple conditional objectives or complicated reliability constraints.

Another approach to RAM optimization that is less computationally expensive is to first optimize the disjoint individual components, then perform a global optimization of the entire RAM. This technique can be less computationally expensive because of its reduced scope. There is also a significant amount of parallelism that can be exploited when using this approach. The results of this type of optimization, however, may not be as accurate as the former method.

The PUMA RAM compiler creates a near-optimal power-delay curve for an entire RAM in two steps. First, near-optimal power-delay curves are created for each buffer leaf cell in the RAM, including the row and column address and write enable buffers. A power-gain curve, which is similar to a power-delay curve, is created for the sense amplifier. Once these cells have been individually optimized, it is possible to optimize the entire RAM by making incremental adjustments to single transistors or by ascending a

previously optimized leaf cell near-optimal power-delay or power-gain curve. A design description that corresponds to any point along the RAM near-optimal power-delay curve can be created by recording the transistor sizes during the optimization process.

There are both advantages and disadvantages to this approach. The primary advantage is that good results can be produced in a short amount of execution time because the optimization technique is a heuristic method. This approach searches the power-delay design space by incrementally adjusting transistor sizes. Consequentially, the near-optimal power-delay curves produced by this tool are not as smooth as those produced by more exact approaches. The roughness of these curves is worsened by the multiple, competing objectives that exist when optimizing an entire RAM design. A more precise method must be used if a more accurate view of the near-optimal power-delay design space is required. For the purposes of design rule cost/benefit analysis, however, this method is adequate.

3.5.1 HSPICE-based Power Gradient Descent

The RAM compiler can use two different optimization techniques to create near-optimal power-delay and power-gain curves for individual leaf cells. The first is based on HSPICE's gradient descent optimization tool which produces a set of transistor sizes for a circuit that meets a set of power, voltage, and timing objectives. This technique requires an initial guess and produces a result that is a local minimum in the region surrounding the initial guess. The optimization algorithm for a buffer cell is as follows:

```
for (delay=MAX_DELAY; delay>0; delay=delay-50ps)
   {
      transistor_sizes=INITIAL_GUESS;

      minimize power dissipation subject to:
         t_pr <= delay,
         t_pf <= delay,      /*Delay constraints*/
         V_OLmax <= VOLMAX,
         V_OHmin >= VOHMIN;   /*Noise margin constraints*/

      if(impossible)
         break;

      logfile[delay]=transistor_sizes;
   }
```

Since the accuracy of the HSPICE's optimization tool is highly dependent on the placement of the initial guess, it is desirable that multiple optimization attempts be made from several different initial starting points. Parallelism can be exploited to reduce the amount of time required to complete multiple optimization runs. This is done by dividing the design space into roughly equal regions, placing an initial guess into the center of each region, and running in parallel an optimization attempt from each initial guess. As the number of available parallel processors increase, the likelihood of finding the global minimum increases.

3.5.2 Efficiency Gradient Ascent

The second optimization method employs an iterative efficiency gradient ascent algorithm that is shown in Fig. 3.16(a). In this algorithm, the transistors are set to initial minimum sizes and a corresponding RAM cell library is generated. The parasitic capacitances and resistances of each cell are extracted and inserted into each cell's SPICE subcircuit model. A representative RAM SPICE file is then generated and simulated to obtain baseline power and delay data. The optimizer then determines the set of incremental transistor size changes that correspond to the greatest increase in efficiency (defined as the most power-efficient reduction in delay). Once the most efficient direction has been determined, the new transistor sizes with their accompanying change in power and delay are recorded so a near-optimal design description can be created for that design point. This algorithm continues until it is no longer possible to reduce delay. Fig. 3.16(b)

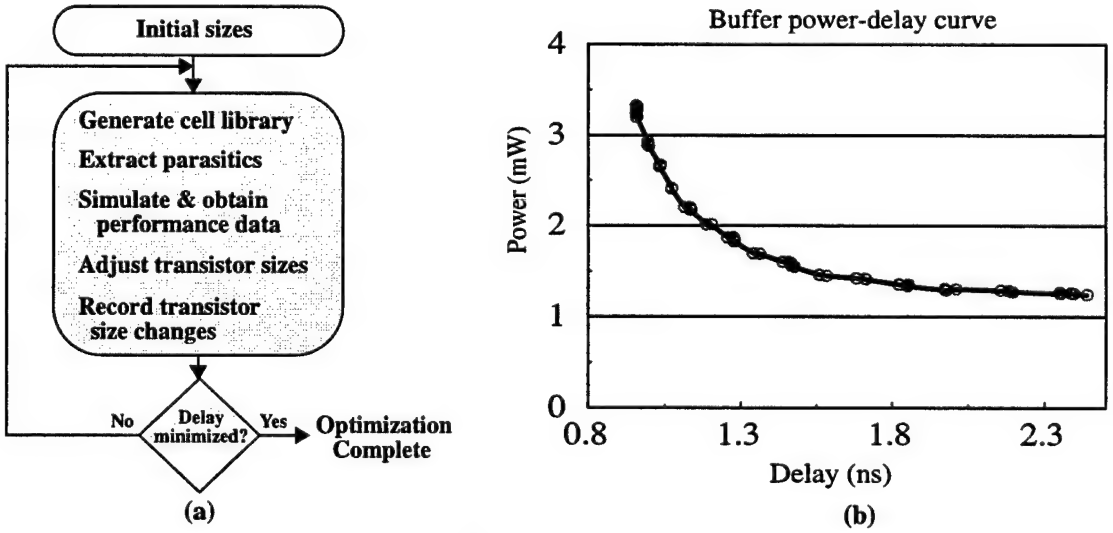


Fig. 3.16 Iterative efficiency gradient ascent algorithm.

gives the results of a buffer optimization using this algorithm. Each circle on the power-delay curve corresponds to a point for which a set of transistor sizes is available.

Fig. 3.17 is a flow chart that illustrates the efficiency gradient ascent transistor sizing algorithm. As previously mentioned, determining which transistor size changes produce the most efficient reduction in delay is done by incrementally adjusting the sizes of the transistors in a cell and observing the resulting change in power and delay. The parallelism inherent to this technique can be exploited by first creating numerous copies of the original cell with different sets of transistor sizes and simulating them concurrently on separate processors. When the simulations have finished, the complete set of cell versions with their accompanying effects on power and delay are observed. For all versions that reduce both power and delay, a corresponding efficiency score is calculated as follows:

$$\text{Efficiency}_a = |\Delta\text{Power} \cdot \Delta\text{Delay}| \qquad \text{Where:} \quad \Delta\text{Power} = \frac{\text{Baseline power}}{\text{New power}} \quad \Delta\text{Delay} = \frac{\text{Baseline delay}}{\text{New delay}} \qquad (3.1)$$

The version with the highest efficiency score is chosen and the optimization algorithm continues. If there are no versions that reduce both power and delay, then those versions that decrease delay but increase power are considered and a corresponding efficiency score is calculated as follows:

$$\text{Efficiency}_b = \frac{|\Delta\text{Delay}|}{|\Delta\text{Power}|} \qquad (3.2)$$

The version having the highest efficiency score is then chosen and the optimization algorithm proceeds. If there are no version that reduce delay, the algorithm enters lookahead mode. In this mode, each version forms the basis for a second set of version that are then simulated and checked for reductions in delay. If one is found, the most efficient direction is selected and the algorithm returns to normal operation. If none of the second-generation versions reduce delay, the algorithm enters lookaround mode. In this mode, the optimizer backtracks to a previous decision and enters lookahead mode from each previously rejected version. If this operation fails to find a reduction in delay, the algorithm terminates. Lookahead and lookaround mode allow the transistor size optimizer

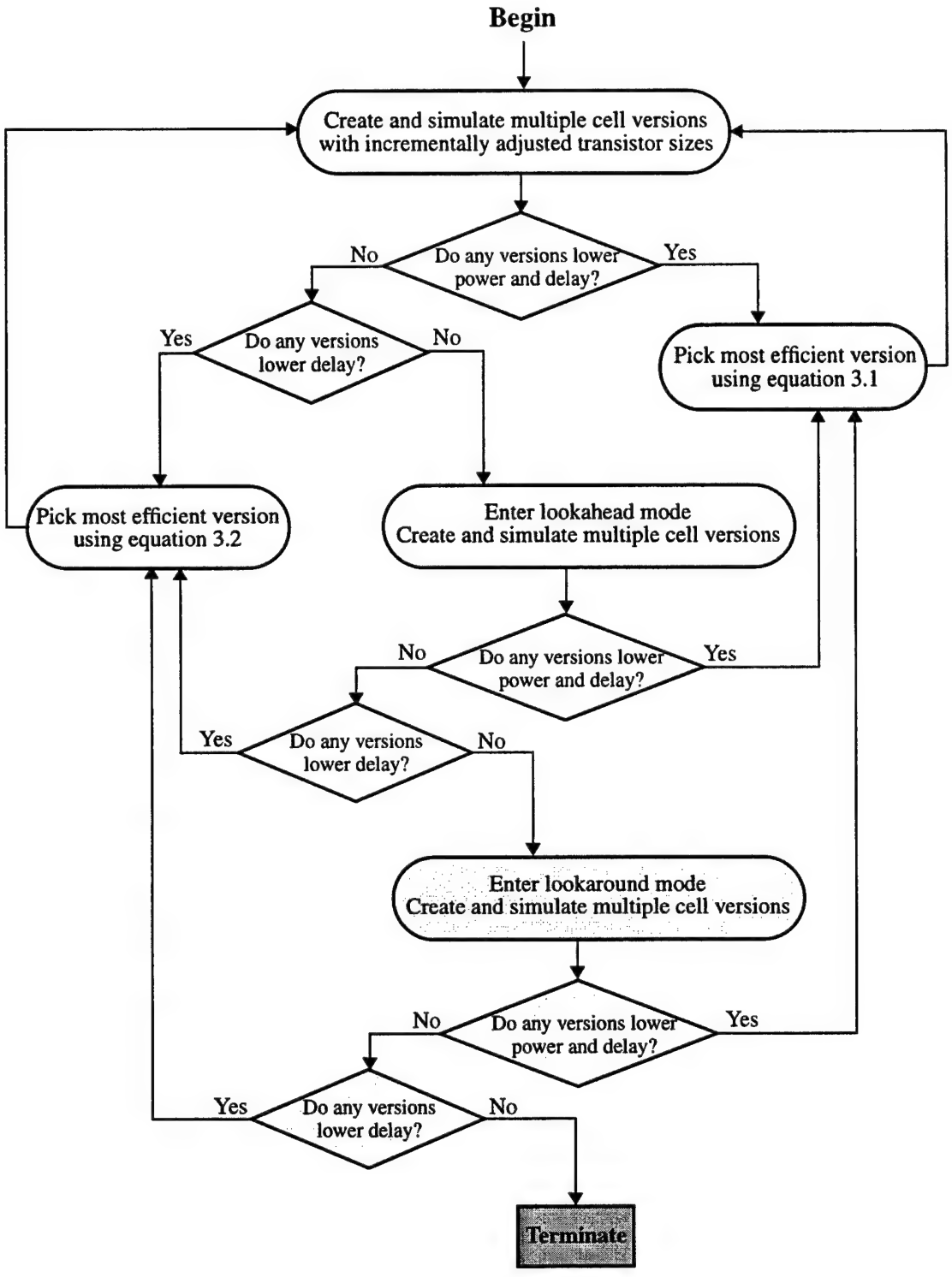


Fig. 3.17 Efficiency gradient ascent transistor sizing flow chart

to "see beyond" local efficiency maximums and increase the probability of finding the global maximum.

3.5.3 Individual Component Optimization

During the first phase of the RAM cell library optimization, a near-optimal power-delay curve is created for the row and column address and write enable buffers. A near-optimal power-gain curve is also created for the sense amplifiers. Either the power gradient descent or the efficiency gradient ascent algorithm can be used to find the near-optimal power-delay curves for the address and write enable buffers. The PUMA RAM compiler allows the user to make this choice. Fig. 3.18 illustrates two near-optimal power-delay curves for a buffer cell. As indicated by the curves, the power gradient ascent algorithm is capable of sizing the transistors for more efficient operation at higher speeds, and the efficiency gradient ascent algorithm is more efficient at lower speeds.

The optimization of a buffer is accomplished by first determining whether the rise or fall propagation time is longest, then finding the most power efficient way to improve that particular propagation time. Fig. 3.19 illustrates this process for a row address buffer. At the beginning of the optimization process, t_{pr} is much greater than t_{pf}. Therefore, the optimizer concentrates on reducing t_{pr} in the most power efficient manner; t_{pf} also changes as transistors are sized to improve t_{pr}. Eventually, t_{pr} becomes smaller than t_{pf}, and the optimization algorithm adjusts, reducing t_{pf} in the most power efficient manner. The

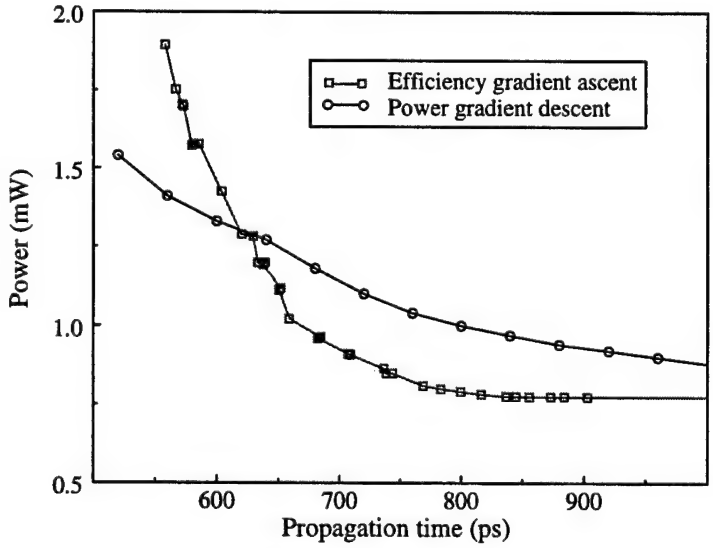


Fig. 3.18 Buffer cell near-optimal power delay curves.

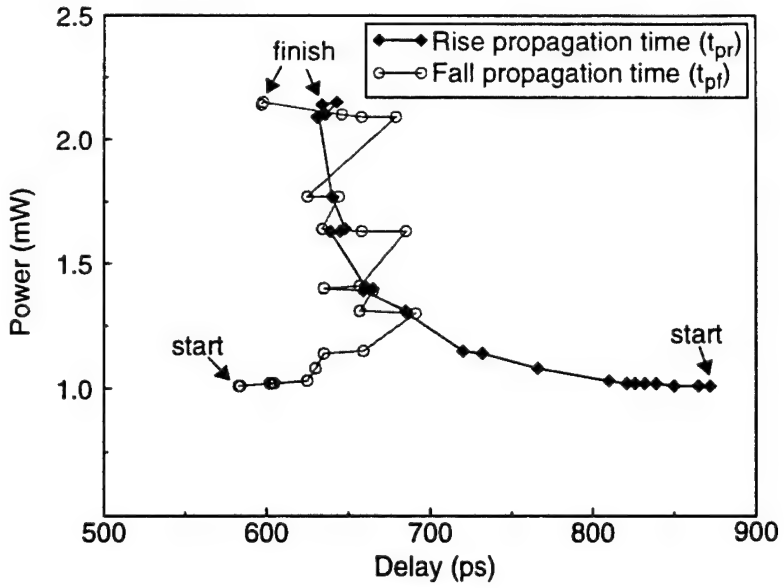


Fig. 3.19 The optimization of a row address buffer.

algorithm terminates when it is no longer able to reduce the larger of t_{pr} and t_{pf}, which, in the case of Fig. 3.19, is t_{pr}.

The optimization of the sense amplifiers is accomplished in a similar manner, however, gain is being maximized instead of delay being minimized. Producing a near-optimal power-gain curve for the differential voltage amplifier of Fig. 3.8 is accomplished by finding the most power efficient manner to increase gain while ensuring that a user-defined V_{OLmax} is not violated. The optimization algorithm first determines V_{OL}. If it is sufficiently low, the optimizer will increase the gain in the most power efficient manner. Once V_{OL} exceeds V_{OLmax}, the optimizer adjusts by concentrating on reducing V_{OL} until it falls below V_{OLmax} before proceeding to increase the gain. Fig. 3.20 illustrates the optimization of a differential voltage sense amplifier.

3.5.4 Synchronous RAM Optimization

Following the creation of near-optimal power-delay and power-gain curves for the RAM buffer and sense amplifier cells, it is possible to begin optimizing the entire RAM as though it were a single cell. The operation of a single cycle synchronous RAM can be divided into the four supporting operations given in Fig. 3.21. For a read operation, the row address lines are driven and decoded during the first phase of the clock cycle. During the second phase, the corresponding word line is asserted and the memory array data

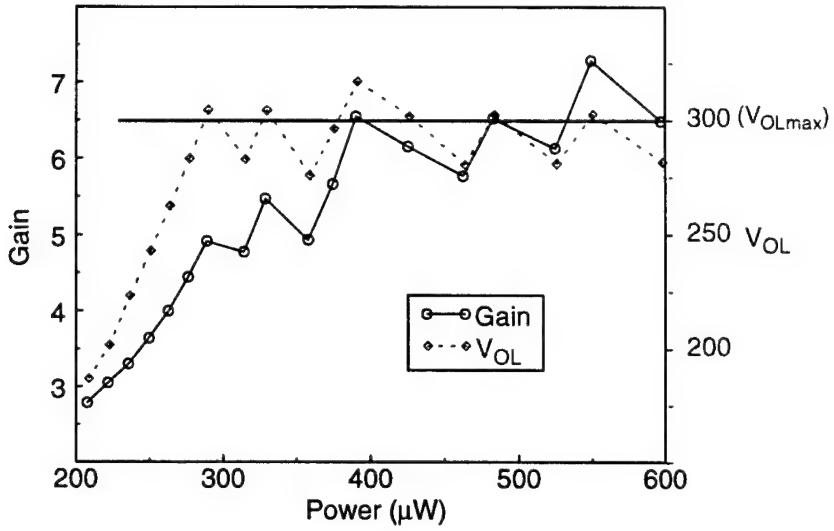


Fig. 3.20 The optimization of a differential voltage sense amplifier.

values are driven onto the bit lines. The sense amplifiers detect the differential voltage on the bit lines and produce a corresponding output.

For a write operation, both the row and column address lines are driven and decoded during the first phase of the clock cycle. At the onset of the second phase, the corresponding word and column buffers are asserted, which writes the desired data into the memory array. The minimum cycle time of a single cycle synchronous RAM will therefore be determined by one of four operations: row address decode, column address decode, data sensing and amplification, or data writing.

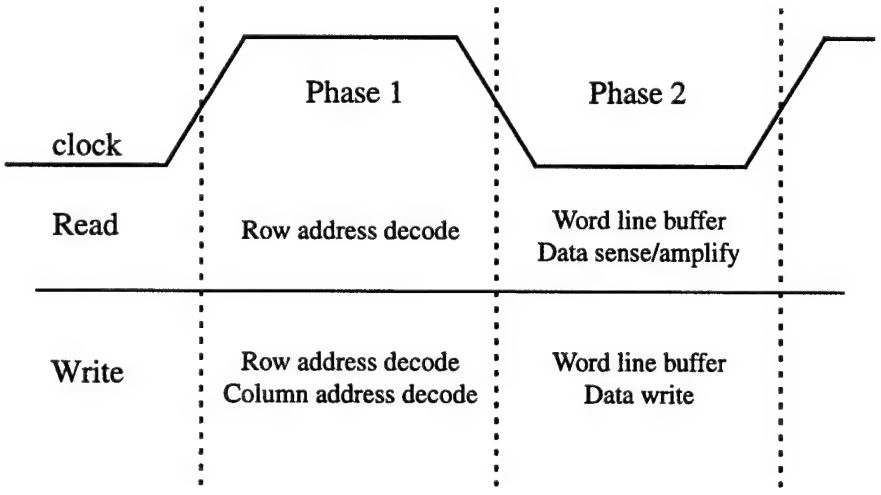


Fig. 3.21 The major operations of a synchronous RAM.

If, for example, a 50% duty cycle is assumed and the data sense/amplify operation requires 2.0 ns to complete while the other three operations require only 1.8 ns, the minimum cycle time will be 4.0 ns. Therefore, minimizing the cycle time of a synchronous RAM is accomplished by iteratively identifying which of the four operations has the largest associated delay and resizing the transistors involved to reduce this delay. The PUMA RAM compiler reduces the delay of the longest operation using the iterative efficiency gradient ascent algorithm. As transistor sizes are adjusted and the associated performance improvement is recorded, a near-optimal power-delay curve is created for the entire RAM.

The optimization of an entire RAM is accomplished by first setting all transistors to their default minimum sizes. A RAM cell library that implements this configuration is then created and extracted to form a representative annotated SPICE netlist. SPICE simulations are then executed to determine which of the four major operations (given in Fig. 3.21) requires the most time to complete. The efficiency gradient ascent algorithm is then used to incrementally adjust the sizes of the transistors that affect that operation. This procedure repeats until it is no longer possible to reduce the delay associated with the speed-critical operation.

Fig. 3.22 illustrates the RAM optimization results for a 2 k-byte RAM implemented in a 0.5 μm CGaAs process. Each line in the plot represents one of the four synchronous RAM operations described in Fig. 3.21. The cycle time for the RAM at a particular power dissipation level is equal to twice the delay associated with the slowest operation at that point. As the optimizer designs RAMs which expend more power, the delay associated with the operation in the critical path is reduced until an improvement in delay is no longer possible.

The roughness of the power-delay curves comes as a result of the granularity used in incrementally adjusting transistor sizes. The jagged peaks in the curves come as a result of the optimizer attempting to meet multiple competing objectives and constraints during the optimization process. When a particular constraint or objective adversely affects another constraint, the optimizer may make several transistor size adjustments to satisfy

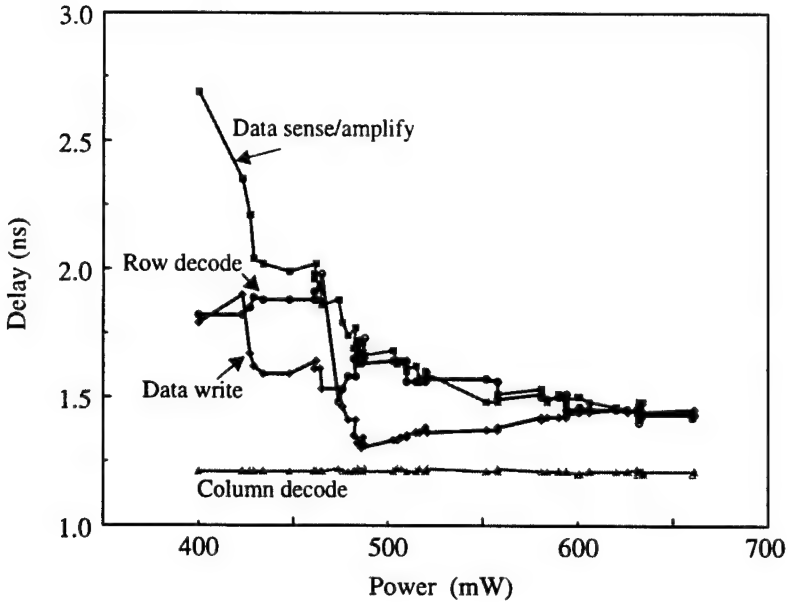


Fig. 3.22 Optimization of a synchronous RAM.

them. During this process, the overall efficiency of the RAM may be reduced in order to ensure correct operation.

Large vertical movements in the row decode operation come as a result of the row address buffers being adjusted in large steps. Since there are not many row buffers in a RAM, their power dissipation does not have a significant impact on overall RAM power dissipation. By adjusting these buffers in larger steps, the optimization execution time is significantly reduced.

The near-optimal power-delay curve for the entire RAM can be obtained from Fig. 3.22 by plotting power versus twice the maximum delay of the four synchronous RAM operations. Fig. 3.23 is the near-optimal power-delay curve associated with the 2 k-byte CGaAs RAM of Fig. 3.22. As mentioned previously, the discontinuities in the power-delay curve are created when a non-timing critical path operation is adversely affected by a transistor size adjustment. When this occurs, the adversely affected operation can become the new timing-critical path with a longer delay than the previous timing-critical operation.

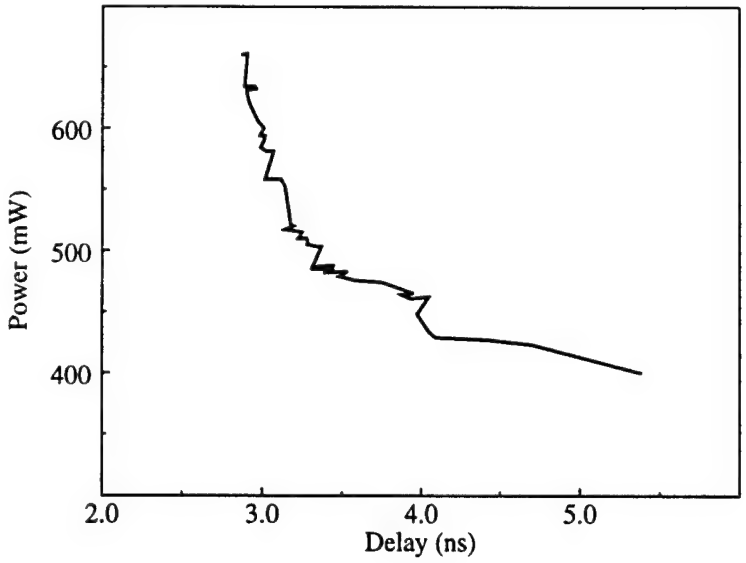


Fig. 3.23 Near-optimal power-delay curve.

3.5.4.1 Address Decoder Optimization

Fig. 3.24 indicates the transistors and elements which are considered for adjustment when reducing the row address buffering and decoding delay. The row address buffer is tuned by traversing its near-optimal power-delay curve, which was created prior to global RAM optimization. The pulldown NFETs in the static NOR-gate row decoder are left as minimum width transistors because they directly impact the capacitive loading on the row address lines and the row decode node, which needs to be minimized to reduce delay. The widths of the word buffer final stage FETs are considered for adjustment during the optimization of the data read and write operations.

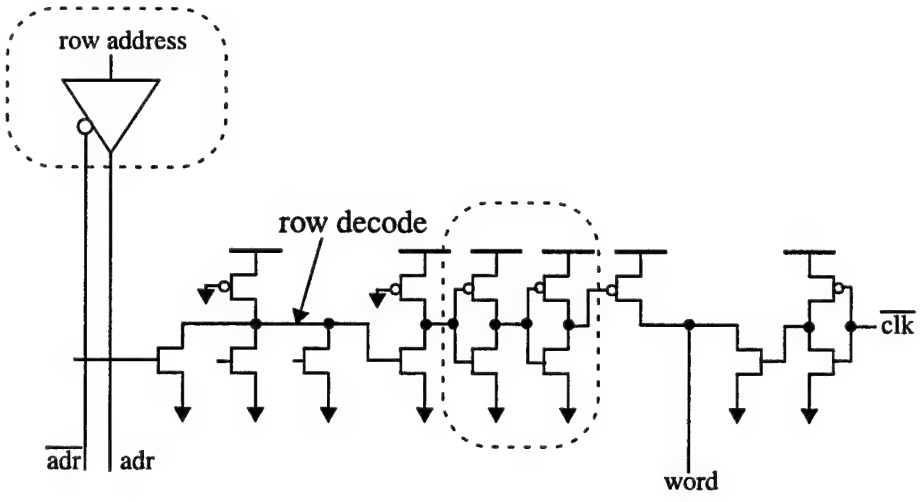


Fig. 3.24 Row address buffering and decoding optimization

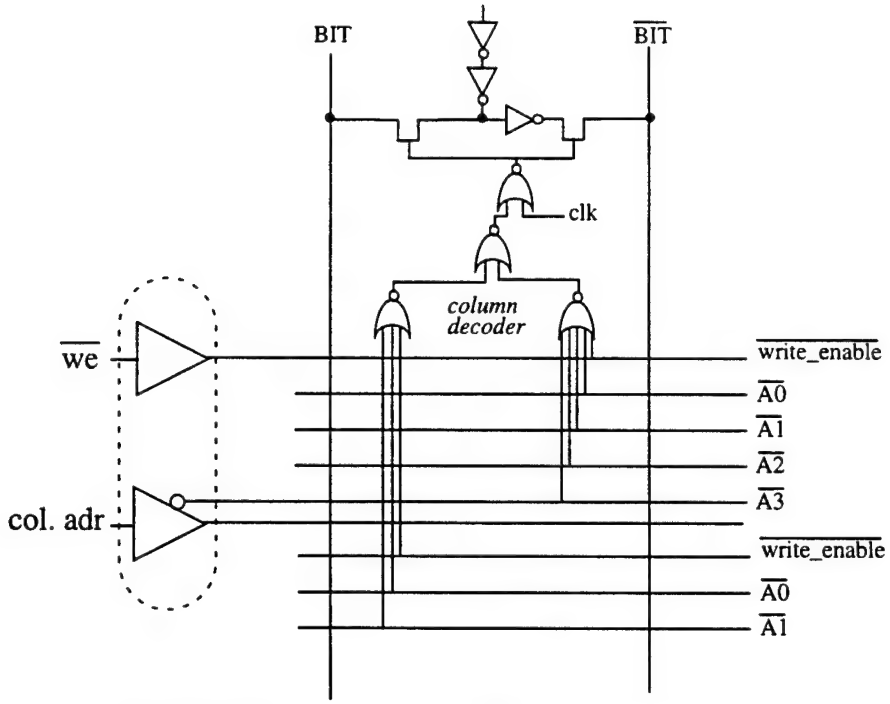


Fig. 3.25 Column address buffering and decoding optimization

Optimization of the column address decoding circuitry is accomplished in a similar manner. Fig. 3.25 indicates which transistors and elements are considered for adjustment when reducing the column address buffering and decoding delay. The column address and write signal buffers are adjusted by traversing their near-optimal power-delay curves. The static NOR gates used for column decoding and write signal enabling are minimally sized in order to reduce loading on the column address, write enable, and clock lines. The write data buffer and tri-state pass transistors are considered for sizing during the optimization of the data write operation.

3.5.4.2 Data Sense and Amplify Optimization

The optimization of the data sense and amplify operation involves tuning the sense amplifier, precharger, word line buffer, and the 6T memory cell. Fig. 3.26 illustrates the transistors and components which are considered for adjustment when optimizing this circuit. The differential amplifier portion of the sense amplifier is tuned by traversing its power-gain curve, which is created prior to global RAM optimization. The strength of the word buffer can be adjusted to increase the rate at which the word line is pulled toward V_{DD}. As the width of pullup PFET in the final stage of the word line buffer is increased,

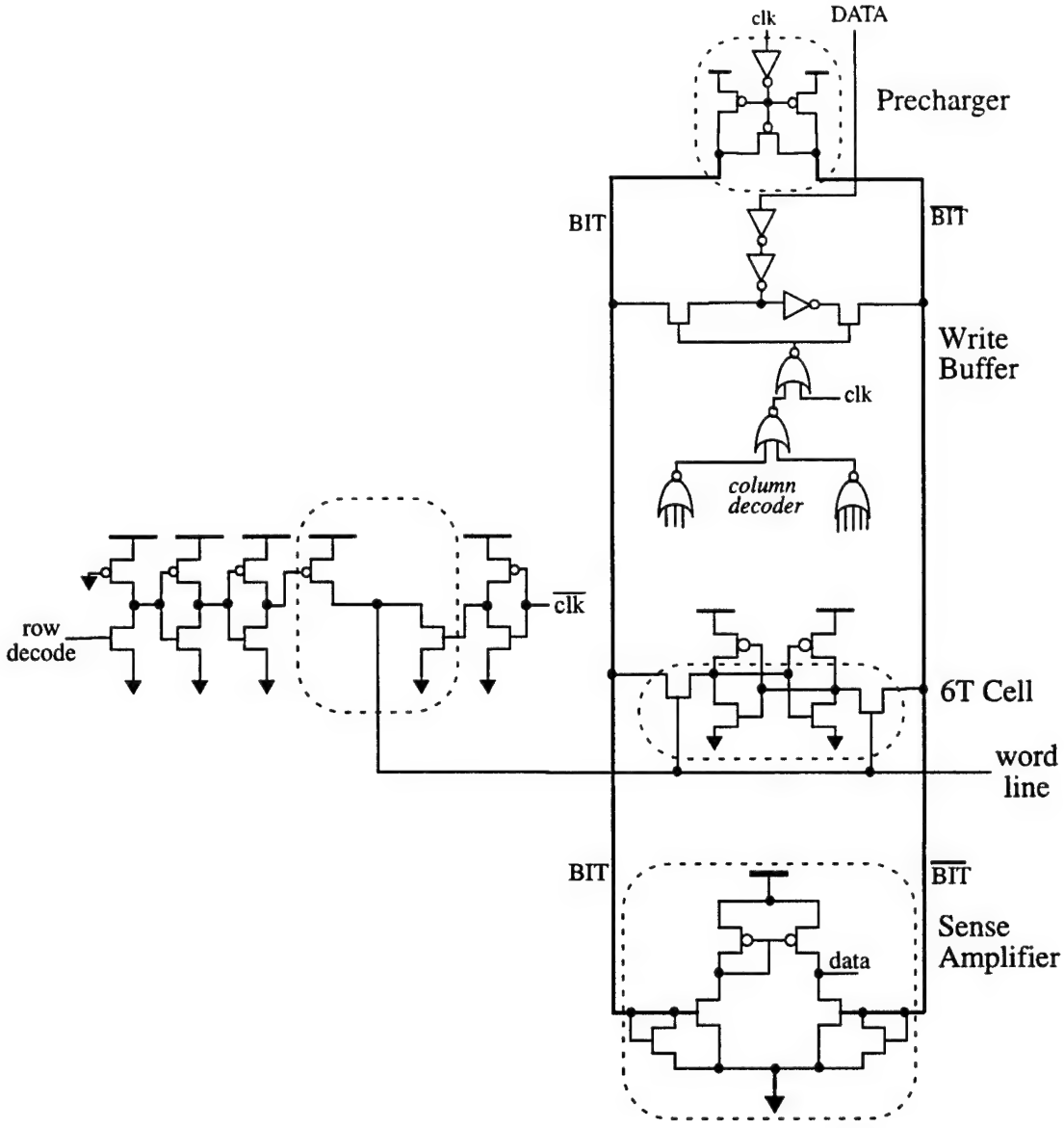


Fig. 3.26 Data sense/amplify optimization.

the width of the corresponding pulldown NFET must also increase to maintain an acceptably low V_{OL} on the word line. The transistors in the precharger can be adjusted to ensure an acceptably small voltage difference on the bit lines at the start of the second phase of the clock cycle. The widths of the bit line pulldown transistors, which are adjacent to the differential voltage amplifier, can be increased to lower the precharge voltage on the bit lines. The pulldown and access transistors in the 6T memory cell can also be adjusted during the optimization of the data sense and amplify operation. The

ramifications of altering the 6T memory cell, and the methodology for ensuring that a stable cell is consistently maintained, are discussed in section 3.6.

3.5.4.3 Data Write Optimization

To reduce the data write delay, the widths of the precharger, word buffer, write buffer, or 6T memory cell transistors are adjusted. Fig. 3.27 indicates the transistors and elements that are considered for adjustment during this process.

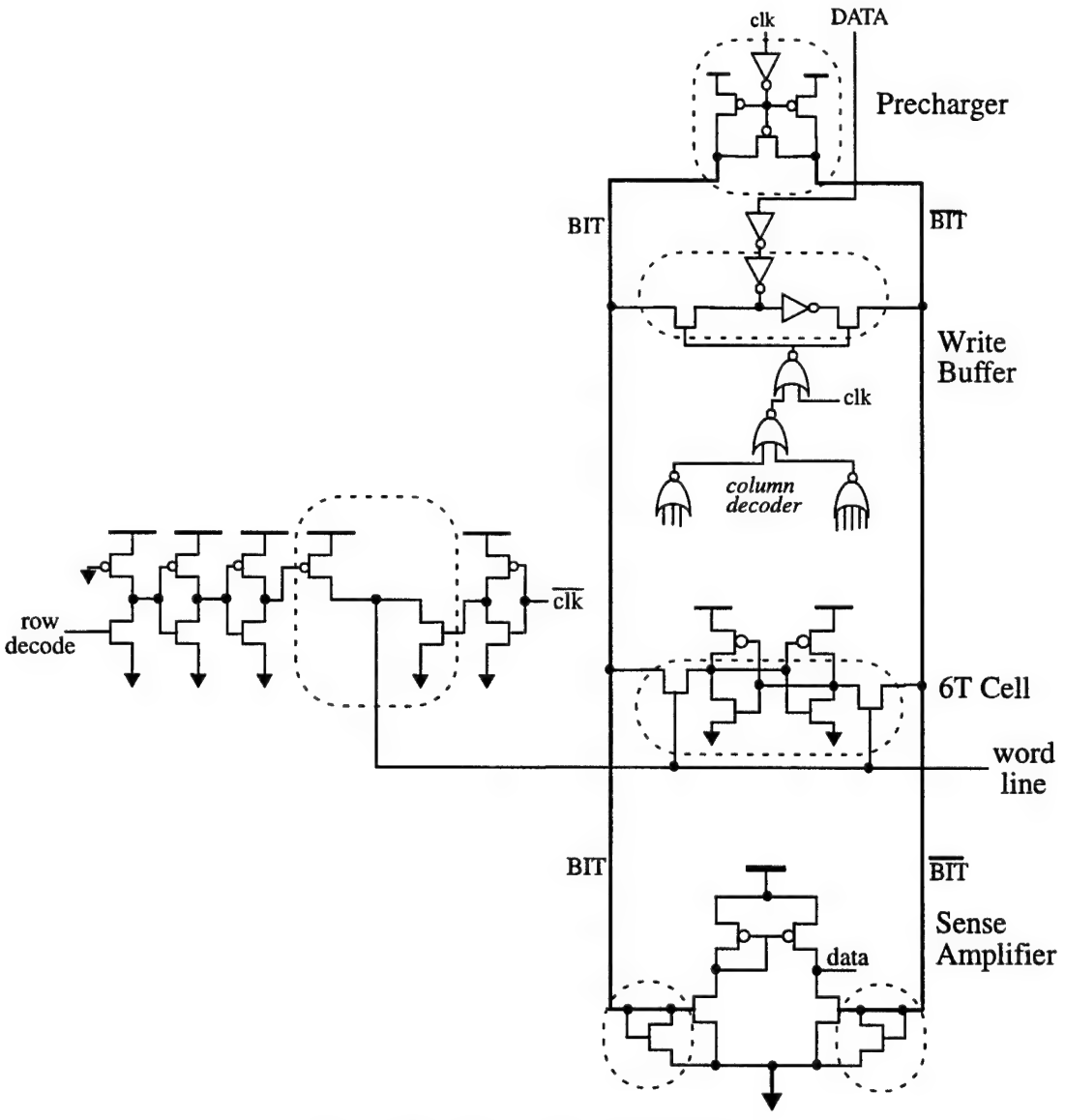


Fig. 3.27 Data write optimization.

3.6 Memory Cell Stability

One of the most critical memory design considerations is cell stability. Bateman defines RAM cell stability as, "the ability of the cell to retain data under all voltage, temperature, and process variations and in the presence of noise or other disturbing events." [152] Designing for stability must include considering the processing environment in which the RAM cell is manufactured and the conditions under which the cell is operated.

There are two major causes of RAM instability. The first is the accidental destructive read. This occurs if the bit lines are biased either too high or too low, or if the transistors in the RAM cell are excessively mismatched due to systematic offsets or random process variations. Second, cell instability can result from a single event upset (SEU) such as an alpha particle absorption. SEUs generate electron-hole pairs that can disrupt the charge stored on a node. If a node voltage sinks or rises too far in a memory cell, the data value being stored in that cell could be altered.

Cell stability is directly affected by the RAM cell transistor sizes, asymmetry within the RAM cell layout, and process variations encountered during manufacturing. Selecting an adequate bit line precharge voltage level minimizes the possibility of destructive reads, but does not eliminate it. A RAM cell can be designed for stability in the presence of process variations by carefully choosing the ratios of the transistors within the memory cell. Of course, minimizing cell layout area is also a primary design objective, and cell stability is increased as certain transistors in the RAM cell are increased. A tradeoff must therefore be made between RAM cell stability and area.

3.6.1 Static Noise Margin Analysis

A 6T SRAM cell can be viewed as a pair of inverters connected to dc offset voltage supplies, as shown in Fig. 3.28. V_{Noise} represents a source of static noise that results from offsets and mismatches due to processing variations. This model does not include dynamic disturbances since static noise margin (SNM) analysis [153-155] does not account for this type of disturbance. The objective of design for stability is to produce a 6T cell with noise margins that are large enough to retain its data value in the presence of the dc offset voltage V_{Noise}.

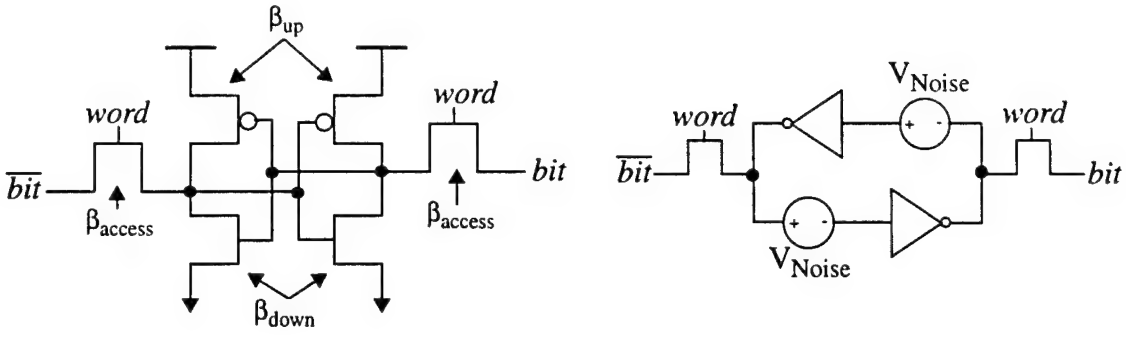


Fig. 3.28 Two views of a 6T SRAM cell.

If the transistors that form the inverters in Fig. 3.28 are identical, the noise margins associated with the cross-coupled inverters can be illustrated with the butterfly plot of Fig. 3.29. In this figure, the inverter transfer curve is superimposed upon itself and the noise margins are shown as a box of maximum area between the superimposed curves. The diagonal distance between the corners of the box give the values of N_{M0} and N_{M1}.

Seevinck et al. analytically derived the value of the SNM for a 6T RAM cell [153], which is given as Equation 3.3. From their derivation, three major conclusions can be drawn. First, the SNM depends only on V_t, V_{DD}, and the cell β ratios (*r* and *q*). Second, the SNM increases as *r* increases, and the SNM is maximized when *r* ($\beta_{down}/\beta_{access}$) and *q*/*r* (or β_{up}/β_{down}) is maximized. Third, the SNM decreases as V_t decreases. When implementing an SRAM in a specific technology, V_{DD} and V_t are constant. For an SRAM cell implemented with minimum width PFET pullups (to minimize layout area), SNM is controlled by the cell ratio *r*.

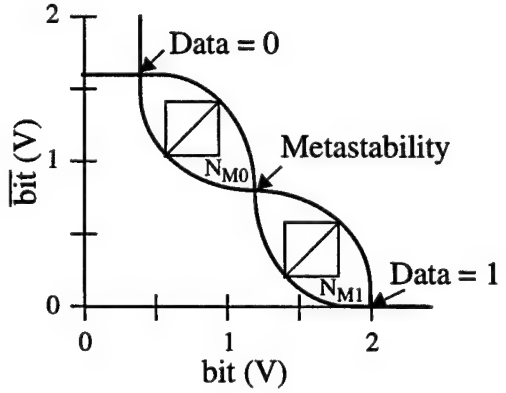


Fig. 3.29 SRAM cell butterfly plot.

$$SNM_{6T} = V_T - \left(\frac{1}{k+1}\right)\left(\frac{V_{DD} - \frac{2r+1}{r+1}V_T}{1+\frac{r}{k(r+1)}} - \frac{V_{DD} - 2V_T}{1+k\frac{r}{q}+\sqrt{\frac{r}{q}\left(1+2k+\frac{r}{q}k^2\right)}}\right)$$

$$r = \beta_{down}/\beta_{access}$$

$$q = \beta_{up}/\beta_{access} \qquad (3.3)$$

$$V_T = \text{threshold voltage}$$

$$k = \left(\frac{r}{r+1}\right)\left(\sqrt{\frac{r+1}{r+1-V_s^2/V_r^2}} - 1\right)$$

$$V_s = V_{DD} - V_T$$

$$V_r = V_s - \left(\frac{r}{r+1}\right)V_T$$

Equation 3.3 gives the SNM for a 6T cell under the assumption of zero systematic offsets and random variations during the manufacturing process. Therefore, the applicability of this analysis is somewhat limited in its scope. To illustrate the effects of systematic offsets, which can result from mask misalignment, Fig. 3.30 gives two versions of a CGaAs 6T SRAM cell. Fig. 3.30(a) gives the mask geometries as drawn, while Fig. 3.30(b) shows the resulting geometries as they are likely to appear on the CGaAs wafer.

The drawn CGaAs 6T cell masks of Fig. 3.30(a) are made of rectangular polygons with sharp corners and appear perfectly aligned with each other. On the wafer, the corners are rounded and the different masks are not perfectly aligned. Fig. 3.30(b) shows a 6T cell where the gate metal mask has been misaligned upward and left of center. As a result of the upward misalignment, the lengths of the left access transistors and the pullup PFETs are slightly larger than their nominal values. Because of the left misalignment, the widths of the right inverter's pullup and pulldown PFETs are also slightly larger than their nominal values. The misalignment of the gate mask therefore produces a 6T cell where the lengths and widths of the transistors are not uniform. The resulting butterfly plot for this 6T cell is given in Fig. 3.31. As illustrated in the figure, N_{M0} is much smaller than N_{M1}, resulting in a cell that is more likely to inadvertently flip from 0 to 1.

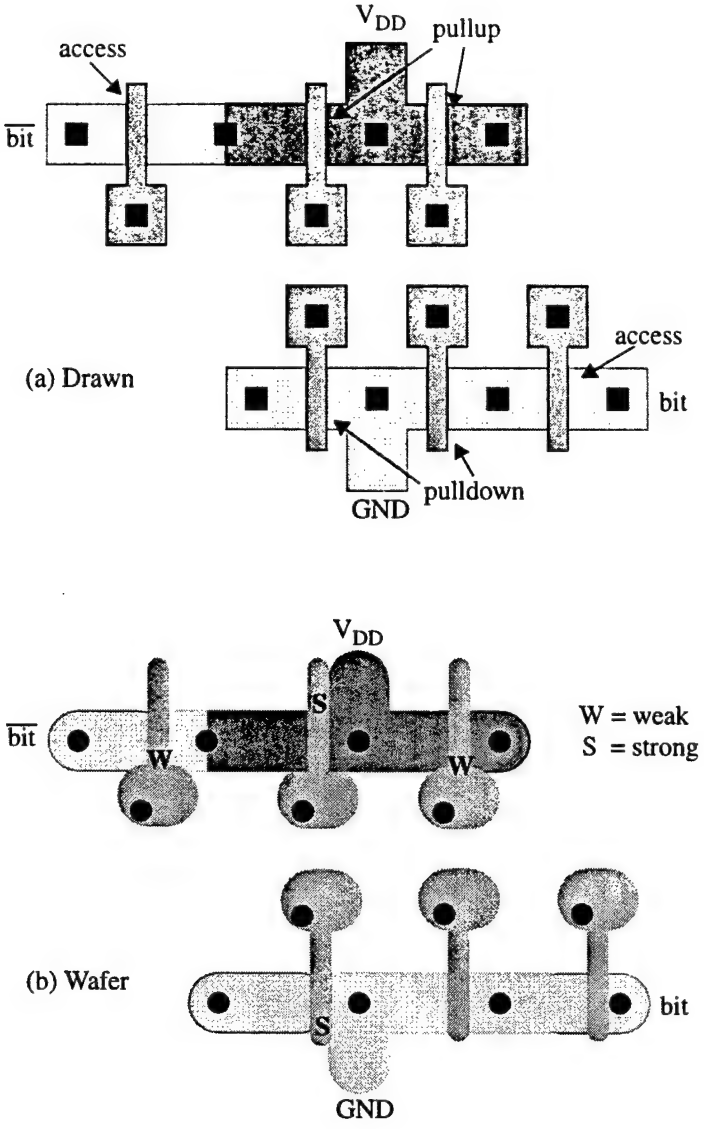


Fig. 3.30 Two views of a CGaAs 6T cell's geometry.

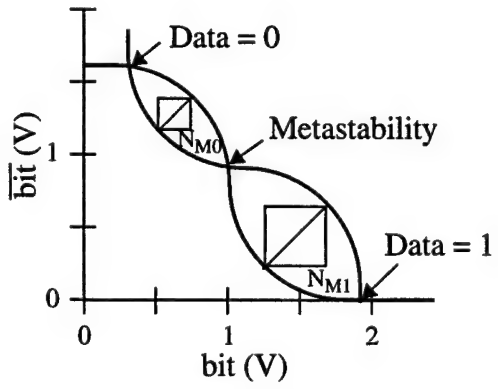


Fig. 3.31 Butterfly plot for a skewed 6T cell.

3.6.2 High Voltage Read Stress Analysis

Static noise margin analysis is useful in illustrating how 6T cell noise margins are compromised due to systematic offsets and/or random process variations. However, it does not provide a useful quantitative measurement of 6T cell stability at a particular operating voltage. If used alone, static noise margin analysis is not adequate for ensuring the stability of a 6T cell. High voltage read stress analysis, on the other hand, does provide this information and is the most common technique for SRAM stability analysis.

The purpose of the high voltage read stress test is to simulate the conditions in the RAM cell that are most likely to create a significant dc offset voltage within the 6T cell. This occurs at the beginning of a read operation, when the word line is asserted and both bit lines are precharged to their maximum voltage level. In a high voltage read stress test, the bit lines, word line, and pullup PFET drains are all connected to the same node, as illustrated in Fig. 3.32. An initial value is written into the SRAM cell, then the common supply node, initialized to a low voltage level, is raised until the data value being stored in the SRAM cell is changed. The voltage level at which the data value changes indicates the operating voltage at which the cell is no longer stable. If this voltage level is at or below the maximum nominal operating voltage of the RAM design, the cell will be unstable and should be modified. Moreover, this voltage level should be comfortably above the planned operating voltage of the RAM design in order to ensure stability in the presence of transient supply overshoots.

High voltage read stress analysis is especially useful in determining the stability of a RAM cell that has been affected by systematic offsets or random process variations. As

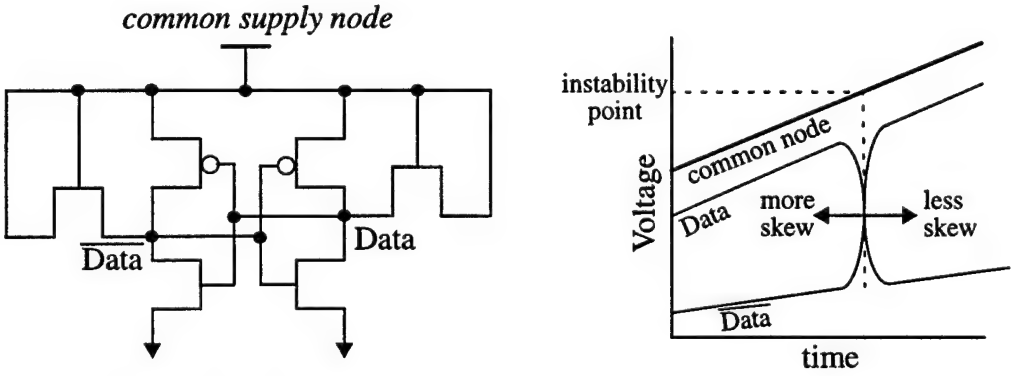


Fig. 3.32 High voltage read stress analysis configuration.

the magnitude of the systematic offset or process variation skew increases between the transistors in the RAM cell, the cell's instability point decreases, as illustrated in Fig. 3.32.

3.6.3 PUMA RAM Compiler 6T Stability Analysis

The PUMA RAM compiler is capable of adjusting the sizes of the transistors within the 6T memory cell during the optimization process. It is therefore crucial that these transistors be altered in ways that ensure cell stability while improving performance. Otherwise, portions of the RAM near-optimal power-delay curve would be invalid due to the instability of the 6T cell.

As previously mentioned, the efficiency gradient ascent algorithm is used to optimize the entire RAM as though it were a single cell. When considering transistor size versions that involve the 6T SRAM cell, the PUMA RAM compiler verifies the performance and stability of the new cell in four steps. First, the cell version is generated in the target process using MasterPort. Second, the parasitic elements of the cell are extracted and an annotated cell netlist is created. Third, the stability of the cell is verified using a high voltage read stress test. Finally, the cell is inserted into the RAM netlist and simulated to check for an increase in performance. If the cell is both stable and provides the most efficient increase in performance (over the other version under consideration), the new cell is adopted and the optimization algorithm continues.

By examining Fig. 3.30, an understanding can be gained of how the CGaAs 6T SRAM cell is affected by systematic offsets, or mask misalignments. When the gate mask is misaligned upward (relative to the active mask), the gate lengths of the left access transistor and the pullup PFETs increase. Similarly, if the gate mask is misaligned downward, the gate lengths of the right access transistor and the pulldown NFETs increase. This occurs because the gate overlap of contact area is pushed down into the active area, resulting in an increase in effective gate length.

In the presence of a left misalignment of the gate mask, the effective widths of the right pullup and pulldown FETs increase as the gate metal encroaches the power supply node. A similar increase in effective width for the left pullup and pulldown FETs is observed if the gate mask is misaligned to the right. When performing a high voltage read stress analysis of the CGaAs 6T cell, it is important to consider a range of misalignment possibilities that represent the best and worst case scenarios for systematic offsets.

Scenario	Misalignment	
	Left/right	Up/down
1	None	None
2	None	Up
3	None	Down
4	Left	None
5	Left	Up
6	Left	Down
7	Right	None
8	Right	Up
9	Right	Down

Table 3.2 Misalignment scenarios.

Table 3.2 indicates all possible gate mask misalignment (relative to the active mask) scenarios that can be simulated for 6T cell stability.

Scenario 1, for example, represents the best case misalignment condition where the gate metal is correctly aligned relative to active. Scenarios 5, 6, 8, and 9, on the other hand, represent worst-case misalignment conditions. The magnitude of these misalignments is determined by the alignment tolerances of the processing equipment.

When conducting a high voltage read stress test, the PUMA RAM compiler executes tests corresponding to the scenarios of Table 3.2. The magnitude of the increase in gate width and length can be set by the user, but a default of 30% is used as a worst-case value. In each simulation, the 6T cell is initialized to a data value, and the common supply node is swept through a range of voltages. The point at which the data value flips is then recorded. The 6T cell with the lowest instability point determines the voltage at which the cell is unstable. The RAM compiler examines this voltage to ensure that it is greater than the user-specified maximum nominal operating voltage before allowing the new cell to be used in a RAM design. Fig. 3.33 illustrates the SPICE simulation results of a CGaAs 6T cell high voltage read stress test. In this example, the instability point was found to be approximately 2.7 V, which is well above the 1.3 V operating point for a CGaAs RAM.

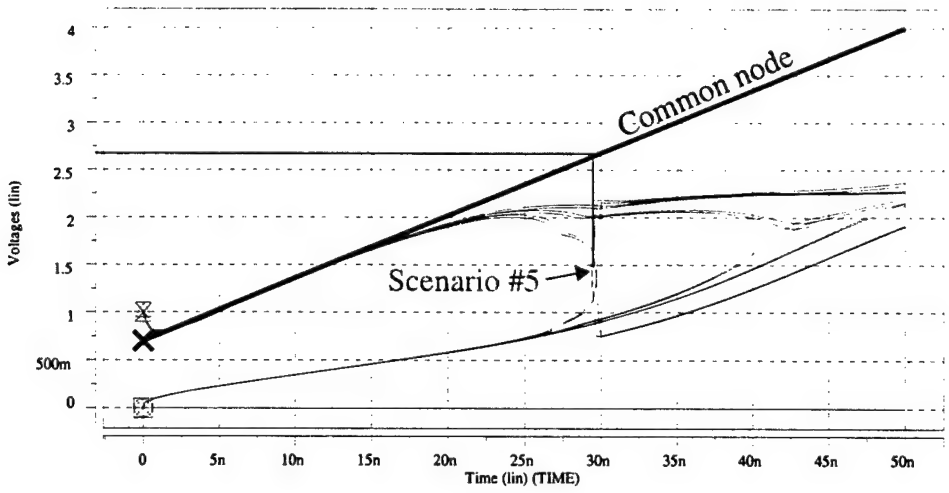


Fig. 3.33 CGaAs 6T cell high voltage read stress test.

3.7 RAM Macrocell Layout Generation

Once a near-optimal power delay curve has been created for a target process, a RAM macrocell can be created for any point along that curve. Identification of a target cycle time specifies a design point along the power-delay curve. The transistor sizes that correspond to that design point are retrieved and used to create a pitch-matched, DRC- and LVS-correct RAM cell library. This is accomplished by evaluating the MasterPort RAM supercell using the retrieved transistor sizes and the target process design rules. Once this cell library has been generated and the power rails have been adequately sized, the desired RAM macrocell can be created by simply tiling the leaf cells according to the user's specifications.

3.7.1 Power Rail Sizing

A RAM macrocell is supplied by two main power grids: the memory core and the peripheral circuit grids. The memory core grid supplies power and ground to the memory cell array, while the peripheral circuit grid supplies power and ground to sense amplifiers, write buffers, prechargers, and row circuitry, as shown in Fig. 3.34. These two grids are connected, so the peripheral circuit grid nets that connect to the memory core grid must be oversized according to the power demands of the memory core.

The main memory array is divided into sections of mirrored 6T cells that are surrounded by major V_{DD} and ground rails. The 6T cells within each section are fed by

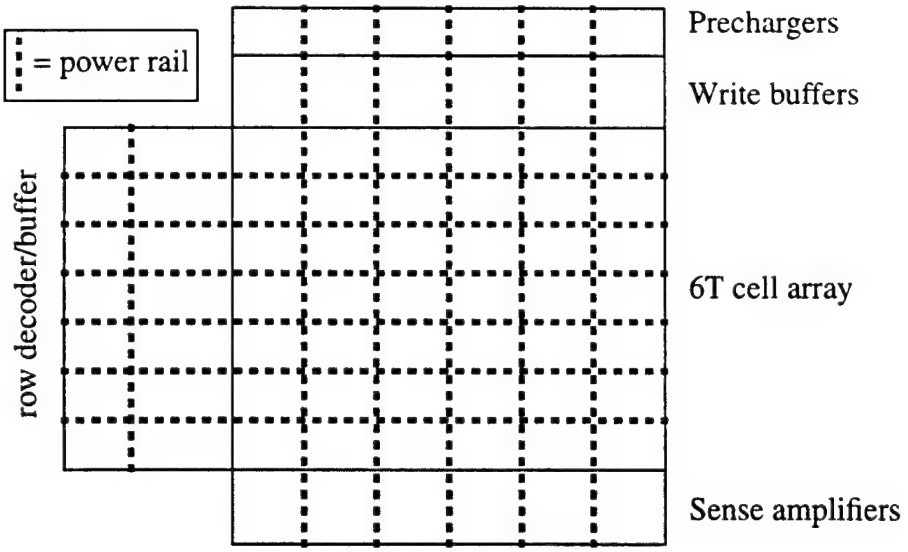


Fig. 3.34 RAM macrocell power grid.

minor power rails that extend horizontally across the section, as illustrated in Fig. 3.35. Fig. 3.36 shows the layout of a memory array section surrounded by its major power rails. The PUMA RAM compiler allows the user to specify the size and aspect ratio of the memory array sections.

Before the final RAM cell library can be generated, the widths of the power rails must be determined. This is accomplished in two steps. First, the rails are sized according

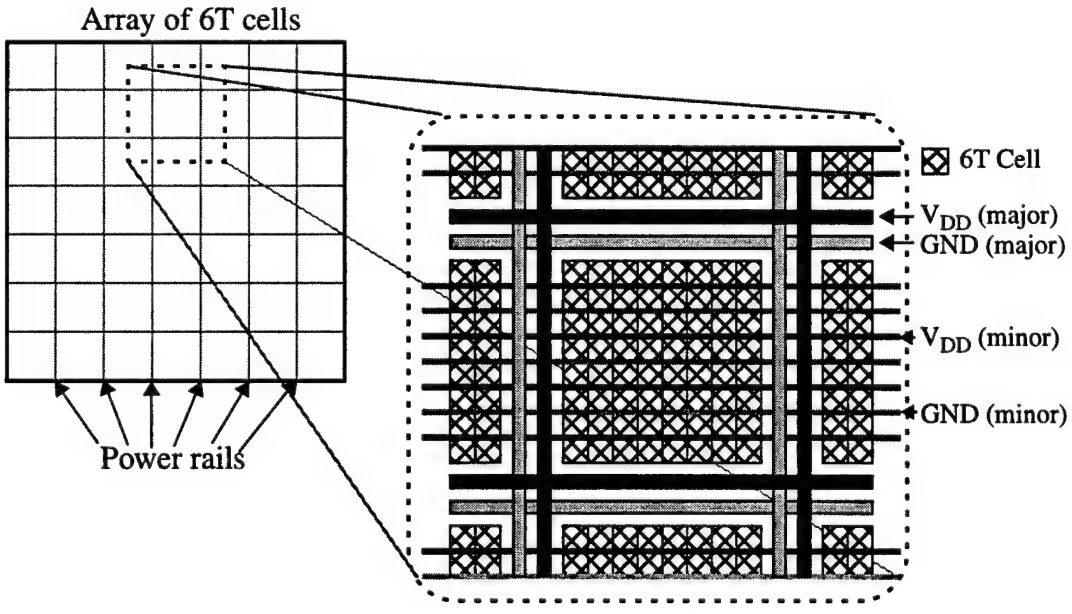


Fig. 3.35 Close-up view of the memory power grid.

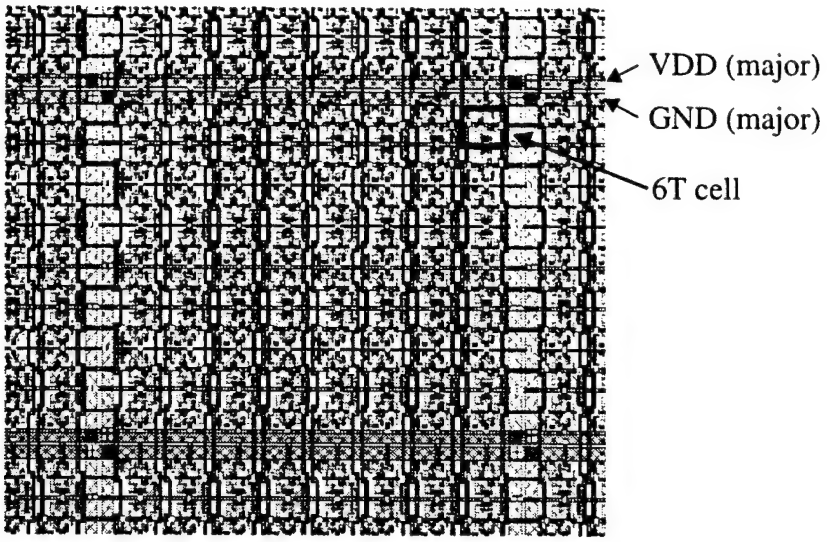


Fig. 3.36 Close-up view of the memory array power grid layout.

to the technology electromigration rules using rms current estimations obtained with SPICE. For example, if a power rail must carry 10 mA (rms) of current and the electromigration rule for that particular layer of metal is 1 mA/μm, the wire must be at least 10 μm wide. Second, an analysis of the power grid is performed to ensure that the minimum ΔV along the major power rails is acceptably low. If ΔV is too high, the rails are widened accordingly. During this type of analysis, peak current estimates are used to determine the worst-case voltage drop values. Peak and rms current estimates are also obtained from SPICE simulation of the RAM at the targeted operating frequency.

Determining ΔV for the major power rails is accomplished by analyzing four different paths through the power grid to the center of the core. The first path enters the macrocell from the top of the layout and proceeds through the prechargers, write buffers, and into the center of the memory array. The second enters from the top of the row decoder array, proceeds down to the center of the row decoder array, then into the center of the memory array. The third enters the macrocell from the bottom of the macrocell and proceeds to the center of the memory core through the sense amplifiers. The fourth enters the memory array directly from the right side. These paths are illustrated in Fig. 3.37.

The widths of the power rails in each path are used in combination with sheet resistance estimates to find the resistances in each path. The peak current requirements of the leaf cells that lie in each path are then determined and used to calculate the maximum

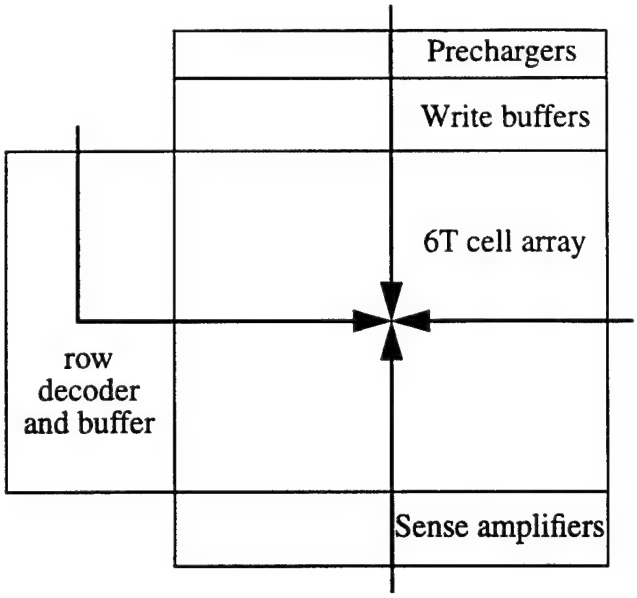


Fig. 3.37 Maximum voltage drop analysis.

voltage drops along each path. If the voltage drops of any of the four paths is greater than the user-specified maximum, the associated power rails are widened and the analysis repeats until the observed voltage drops are acceptably low.

3.7.2 Layout Generation

Once the transistor and power rail sizes are determined, a compacted and pitch-matched RAM cell library is created with MasterPort by evaluating the RAM supercell in the target technology using the specified transistor and rail sizes. The task of instantiating and tiling cells from the library is accomplished with CDS, which is a collection of C system calls that provide access to the cells and geometries that are created by MasterPort. Software that utilize CDS system calls can instantiate and tile MasterPort generated leaf cells into a user-defined RAM configuration. The PUMA CGaAs RAM compiler requires input parameters that indicate the number of memory bit rows and columns to implement, the primary and secondary column decoding ratios, and the number of memory bit cells to be placed between the horizontal and vertical power rails, as shown in Appendix B.

CDS system calls allow leaf cells to be instantiated that are mirrored or rotated versions of the original cells. This allows the PUMA RAM compiler to create two different layout configurations. In one of these, the row decoders are placed on the left side of the memory array, as illustrated in Fig. 3.38. A second, mirrored memory array

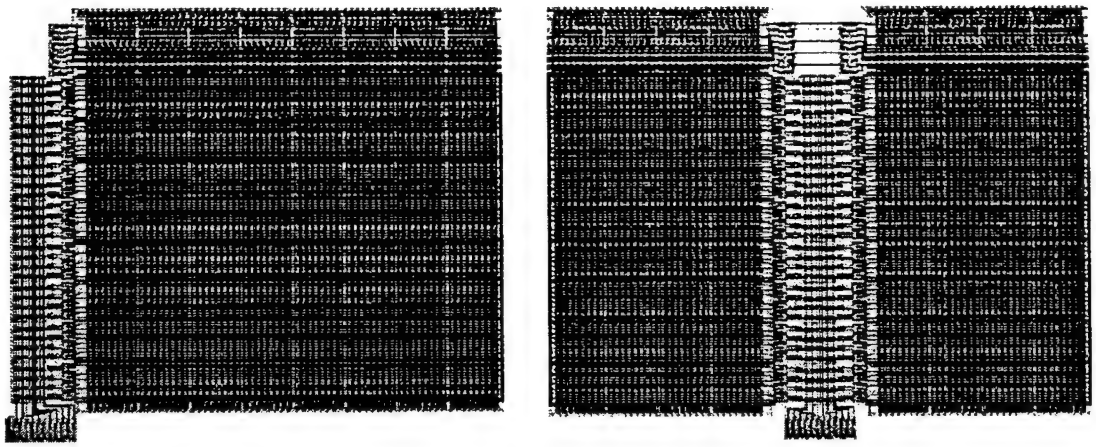


Fig. 3.38 Example 0.5 μm CGaAs RAM macrocell layouts.

can be added, with the row decoders placed in the center. This reduces the loading associated with each word line and allows the row buffers to share a common set of row address decoders. Fig. 3.38 also shows a RAM layout generated in this manner.

As mentioned previously, the RAM compiler supports two write conditions. The primary write signal can be configured to support up to a 16:1 column folding ratio while the secondary write signal can be configured to support up to a 4:1 column folding ratio. These write conditions can be used to support a combination of block-, word- or byte-write capabilities.

After the main RAM layout is generated, including the placement of all necessary peripheral and power rail cells, the NOR-based row and column decoders are encoded to a specific row and column address. This is accomplished by using CDS geometry primitives to place vias onto the decoder leaf cells that connect the decoding NOR gate inputs to either ADDRESS or $\overline{\text{ADDRESS}}$.

Row and column address buffers are instantiated into the design after the RAM layout is complete. These buffers are routed to their respective ports on the row and column decoder array using CDS routing primitives and a simple channel routing algorithm, as shown in Fig. 3.39. Finally, ports are added to the layout which are needed for final layout verification and to integrate the RAM into higher-level designs. The RAM compiler also exports the final RAM layout to GDS2 or CIF formats and creates a corresponding SPICE netlist that can be used for LVS verification.

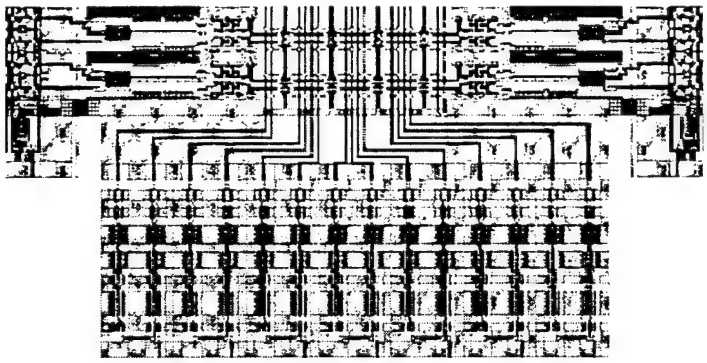


Fig. 3.39 Row address buffers routed to the RAM core.

Another benefit of using the CDS system calls for layout generation is the quick turnaround time they provide. A 4 k-byte RAM layout can be entirely generated in less than ten minutes using a 167 MHz Sun UltraSPARC II.

3.7.3 Example RAM Macrocells

The PUMA RAM compiler has been used to create two RAM macrocells that have been implemented in Motorola's 0.5 μm CGaAs process. The first of these is a 8 x 2048 RAM macrocell with 2 k-bytes of total capacity. Fig. 3.40 shows the near-optimal power delay curve for a RAM of this configuration, and indicates the design point used to create the macrocell. Fig. 3.41 gives the block diagram for this RAM and Fig. 3.42 gives its layout. This RAM measures 3.4 x 2.8 mm and is designed to operate at 333 MHz at 1.3 V; it would dissipate approximately 362 mW at this frequency. The RAM core was implemented as a stand-alone chip for testing purposes, and was not meant to be incorporated into a larger design.

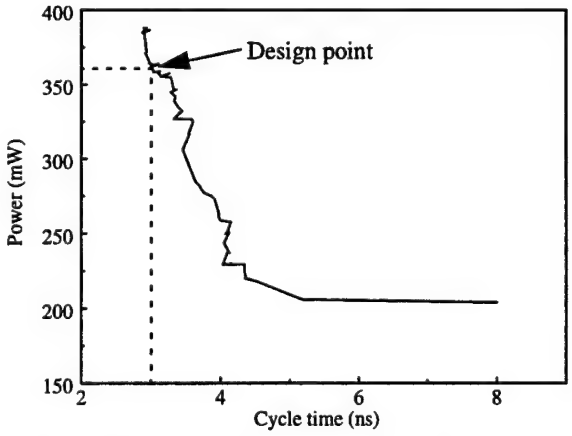


Fig. 3.40 2 k-byte RAM near-optimal power delay curve.

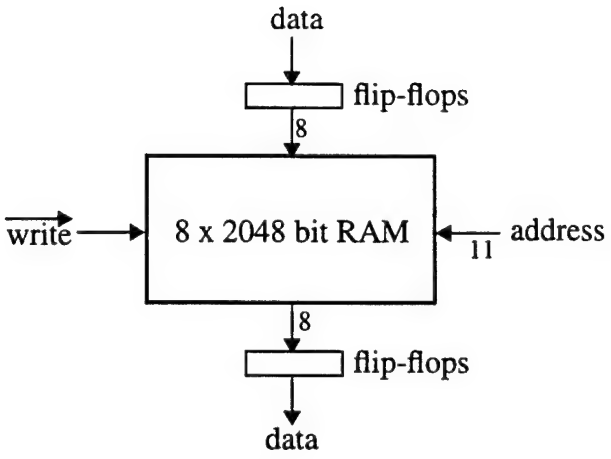


Fig. 3.41 2 k-byte test RAM block diagram.

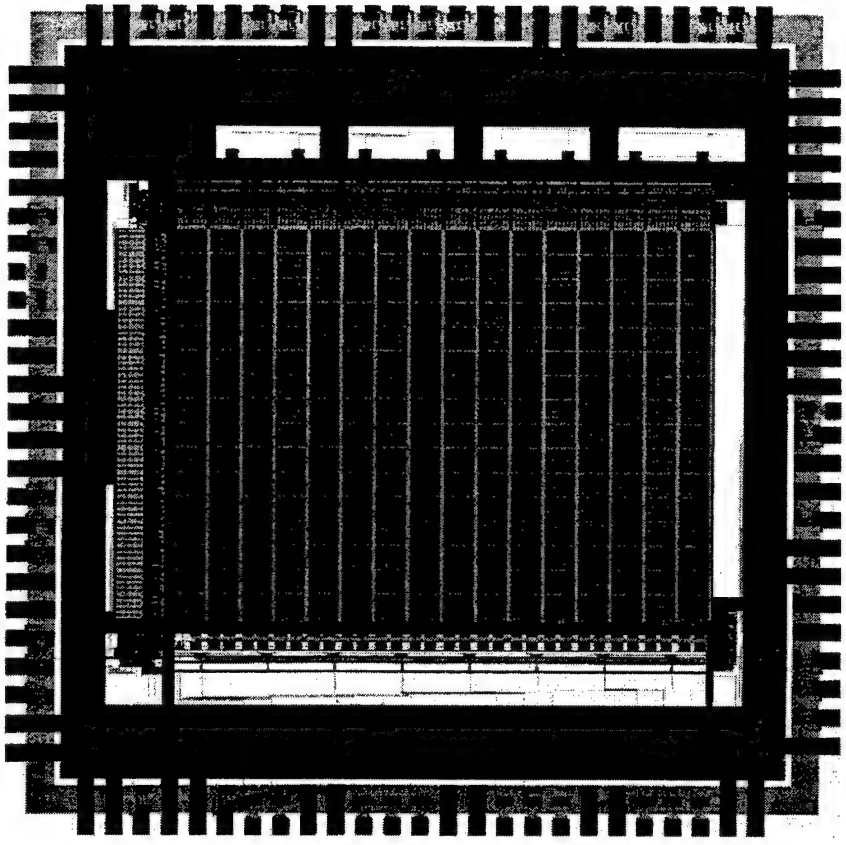


Fig. 3.42 2 k-byte test RAM chip layout.

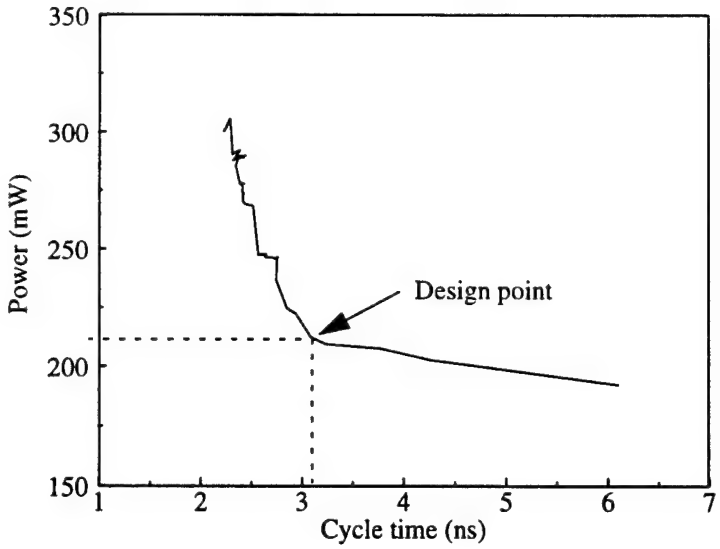


Fig. 3.43 1 k-byte PUMA instruction cache near-optimal power delay curve.

The second RAM design produced by the PUMA RAM compiler is an embedded 1 k-byte instruction cache that is used in the PUMA PowerPC microprocessor. This design implements 64 lines of 128 data bits, 23 tag bits, and a valid bit for a total capacity of 9,728 bits. The near optimal power-delay curve for a RAM of this configuration is given in Fig. 3.43. This RAM measures 4.0 x 1.6 mm, and is designed to operate at a maximum frequency of 320 MHz at 1.3 V, and would dissipate approximately 215 mW at this frequency. Fig. 3.44 gives a block diagram of the instruction cache and Fig. 3.45 illustrates its location within the PUMA PowerPC microprocessor layout.

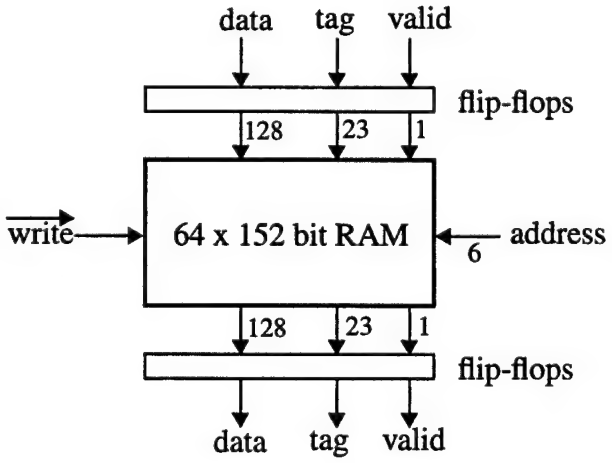


Fig. 3.44 PUMA instruction cache block diagram.

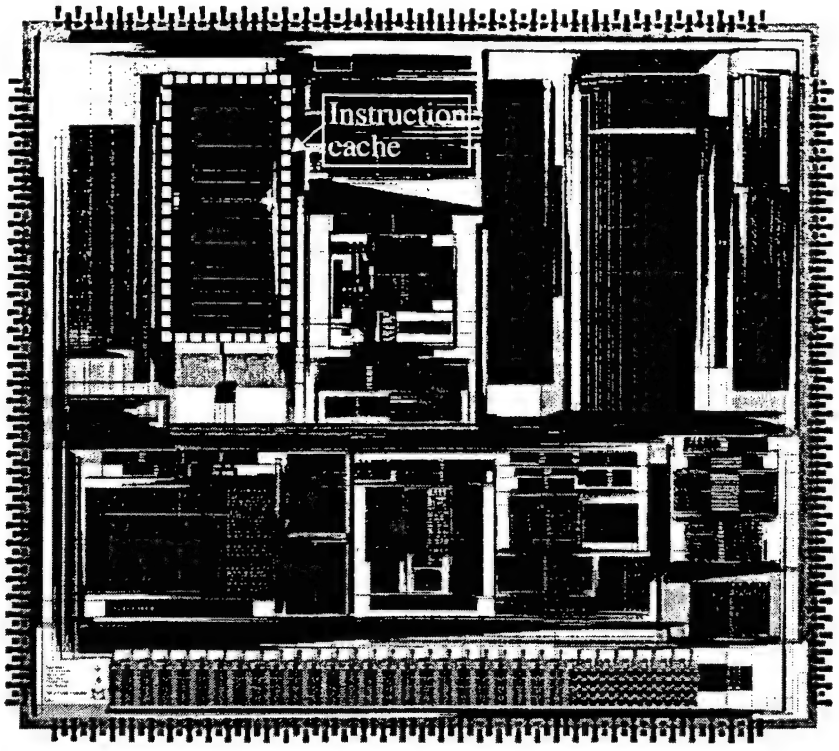


Fig. 3.45 Annotated PUMA PowerPC microprocessor layout.

3.8 Conclusion

A process-independent, optimizing RAM compiler has been designed for use in design rule cost/benefit analysis. This compiler, known as the PUMA RAM compiler, differs from past RAM compilers in several ways. First, it utilizes a constraint graph-based hierarchical layout compactor (Duet Technology's MasterPort) to produce a compacted, pitch-matched RAM cell library for any IC process. Second, it employs an iterative optimization algorithm that explores a large transistor size design space to produce the near-optimal power-delay curve for any particular process. Once this curve has been created, the PUMA RAM compiler can create a DRC- and LVS- correct RAM macrocell layout that meets a user's capacity, organization, and timing specifications.

The RAM compiler is comprised of four major components that access a library of 10 shared C library programs. These components include a cell library generator, a transistor size optimizer, a power rail sizer, and a cell tiling engine. The cell library generator uses MasterPort to compact a two-level hierarchical RAM supercell to produce a pitch-matched RAM cell library for the target process. This supercell instantiates at

least one copy of every unique leaf cell in a way that establishes the pitch-matching and connectivity requirements of the cell library.

The transistor size optimizer uses two optimization algorithms to explore the large transistor size design space. The first is an HSPICE-based power gradient descent algorithm that produces a set of transistor sizes that minimizes power dissipation subject to timing and voltage level constraints. The second is an efficiency gradient ascent algorithm that incrementally searches the transistor size design space for the most power efficient means to decrease delay. The transistor size optimizer first produces near-optimal power delay curves for individual components within the RAM, then uses these curves to optimize the entire RAM as though it were a single cell. This tool produces both a near-optimal power delay curve and a set of transistors sizes that correspond to points along the power-delay curve.

The power rail sizer determines the minimum power rail widths for the RAM in order to provide an acceptable level of immunity from electromigration, and to ensure that maximum voltage drops along the major power rails are acceptably low. This tool first sizes the power rails according to electromigration rules, then verifies that maximum voltage drops are within an acceptable level. If they are not, the power rails are iteratively widened and analyzed until maximum voltage drops are acceptable.

The cell tiling engine utilizes Duet's Compiler Development System to instantiate and tile MasterPort-generated leaf cells. The tiling engine is capable of producing a macrocell layout that meets the user's capacity and aspect ratio specification. It is also capable of implementing up to two write conditions to support combinations of byte-, word-, and block-write conditions.

The RAM core and peripheral circuits were designed to meet several objectives, including robustness and portability to ensure that the design works correctly across a range of IC processes, and minimum delay in order to increase the operating frequency of the PUMA microprocessor. The secondary objectives include minimum circuit area, since CGaAs is a low integration level technology, and low power dissipation. Decoder, buffer, sense amplifier and precharger circuits were designed to meet these objectives.

An essential part of the RAM optimization process is the alteration of the 6T memory cell. Each time this cell is modified in any way, it is essential to check the new

cell for stability. Static noise margin analysis has proven to be an insufficient method, if used alone, for establishing the stability of a 6T cell. The PUMA RAM compiler performs a high-voltage read stress test, which is a more comprehensive technique, on all 6T cells under consideration to determine their stability. Only those cells that are found to be stable are used during the optimization process.

The PUMA RAM compiler has been used to produce two RAMs for Motorola's 0.5 μm CGaAs process. The first is a 2 k-byte stand-alone test RAM that was designed to operate at 333 MHz with a 1.3 V power supply, and is expected to dissipate 362 mW at that frequency. The second is a 1 k-byte embedded instruction cache for the PUMA microprocessor. This RAM was designed to operate at 320 MHz with a 1.3 V supply, and simulation predicts that it will dissipate 215 mW at that frequency. Both of these RAMs have been fabricated, and will soon be tested.

CHAPTER 4

CGaAs COST/BENEFIT ANALYSIS

Motorola's CGaAs process is a relatively new technology that has not enjoyed the extensive development effort of many current CMOS technologies. Even though CGaAs transistor gate lengths have been scaled to 0.5 μm, the remaining horizontal design rules resemble those of a typical 0.7 μm process. In this respect, 0.5 μm CGaAs can be viewed as a process that has been partly scaled, i.e., the transistor gate length has been scaled but the remaining design rules have not. Furthermore, we have cost estimates for scaling each of the CGaAs design rules by up to 40%. For these reasons, CGaAs is an excellent demonstration technology for design rule cost/benefit analysis.

In Chapter 2, a quantitative methodology was described for analyzing IC process design rules from a cost, performance, and area perspective. Through a series of steps, this analysis produces the horizontal design rule shrink proportions that provide an effective tradeoff between these characteristics. First, the area-critical design rules are determined and ranked according to their impact on layout area. Next, these are organized into sets of interdependent design rules which would be scaled as a group. The impact of reducing each of these design rule sets on die cost and performance is then determined. Cost and benefit data is combined to create cost/benefit plots for each of the analyzed design rule sets. The slopes and inflection points of these plots can be used to guide the process engineer in selecting design rule reduction ratios that make an effective tradeoff between cost and performance.

Chapter 3 described a process-independent RAM compiler that can be used to make fair and comprehensive comparisons between processes having different design rules. This tool generates near-optimal power-delay curves to indicate a range of achievable delay values with corresponding minimized power dissipation. The compiler

also produces layout for any RAM on the near-optimal power-delay curve, allowing accurate layout area data to be obtained.

Having defined a design rule cost/benefit analysis methodology, and having verified that the necessary CAD tools for conducting performance and area improvement evaluations are available, it is now possible to perform a design rule cost/benefit analysis using CGaAs as a demonstration technology. This analysis can be viewed as a precursor to completing the 0.7 to 0.5 μm scaling efforts, and therefore does not include gate length reduction. Moreover, gate length reduction analysis is not currently possible because the existing device model is only accurate for 0.5 μm transistors. Models for transistors that have been scaled beyond 0.5 μm have not been made available.

Section 4.1 describes, quantifies, and ranks the CGaAs area-critical design rules for a representative RAM layout. In section 4.2, CGaAs design rule scaling cost estimates are given and used to form the area-critical design rule sets. Section 4.3 presents the area and performance impact analysis. Area improvement data is then used to calculate die cost estimates and the associated cost/benefit plots are formed and interpreted in section 4.4. The results of a CGaAs transistor threshold voltage cost/benefit analysis is given in section 4.5. Conclusions are made in section 4.6.

4.1 Area-Critical Design Rules

The set of area-critical design rules, and their associated impact on layout area, will vary according to the representative layout being examined. For the purposes of CGaAs design rule cost/benefit analysis, a 2 k-byte RAM with a square aspect ratio is used as the representative layout. This RAM configuration reaches an effective compromise between signal wire loading and capacity, and can therefore be duplicated to create efficient higher capacity RAMs. Fig. 4.1 shows the layout of a 2 k-byte RAM that was generated by the PUMA RAM compiler and is used for CGaAs design rule cost/benefit analysis. This RAM is organized as 128 × 128 bits, and supports both block- and word-write capabilities.

Area-critical design rules, when reduced, are the design rules that are most likely to provide a significant improvement in both area and performance. Section 2.3.1 describes a method for finding the area-critical design rules for a particular layout. This

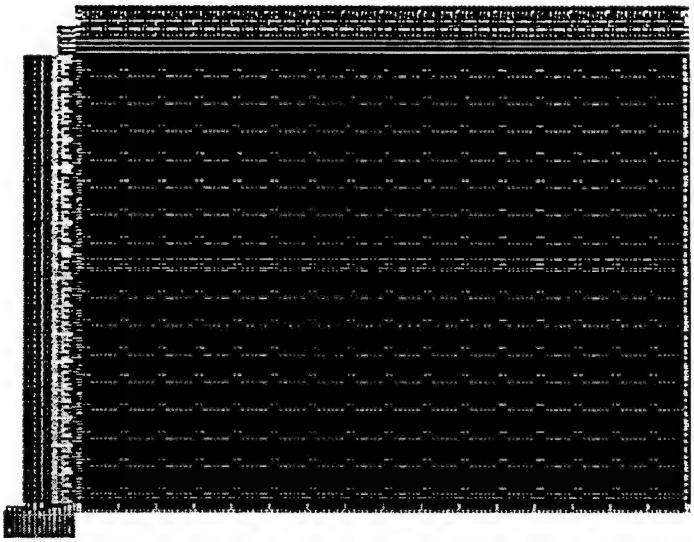


Fig. 4.1 Representative 2 k-byte RAM layout.

method, however, would be extremely time consuming if performed manually. To speed up this process, MasterPort's x- and y-dimensional area-critical path analysis tool is used to indicate exactly which leaf cells and design rules form the area-critical path through a given layout.

The first step in finding the area-critical design rules is to create a representative RAM layout in the baseline process. From this layout it is possible to determine which leaf cells appear in the x- and y-dimensional area-critical path. For the y-dimension area-critical path, each row of cells is examined to determine which leaf cell in that row is responsible for setting the minimum row height. Similarly, the x-dimension area-critical path leaf cells are found by examining each column to determine which cell is responsible for setting minimum column width. MasterPort's critical path analyzer is capable of performing this task on the top level cell of a hierarchical design. This analysis was performed on the 2 k-byte 0.5 μm CGaAs RAM macrocell of Fig. 4.1. The number of occurrences of each leaf cell in the x- and y-dimensional area-critical paths are given in Table 4.1.

Once the x- and y-dimensional area-critical path leaf cells have been identified, it is necessary to identify and quantify the area-critical design rules within each of these cells according to the method described in section 2.3.1. MasterPort's critical path analysis tool is used for this purpose. Fig. 4.2 illustrates a y-dimension area-critical path

Leaf cell	Number of occurrences	
	x-critical path	y-critical path
6T memory bit	128	128
Sense Amplifier	0	1
Write buffer	0	1
Precharger	0	1
Word line buffer	1	0
Row decoder	4	0
Horizontal power rail	0	18
Vertical power rail	18	0

Table 4.1 Leaf cell occurrences in the area-critical paths.

analysis of a 6T memory cell pair. The dark lines through the layout indicate the design rules that form the y-dimension critical path.

Table 4.2 gives the 12 highest ranked area-critical design rules, with their corresponding number of occurrences in the RAM's x- and y-dimensional area-critical paths, the total number of occurrences, and the normalized score. The actual design rule

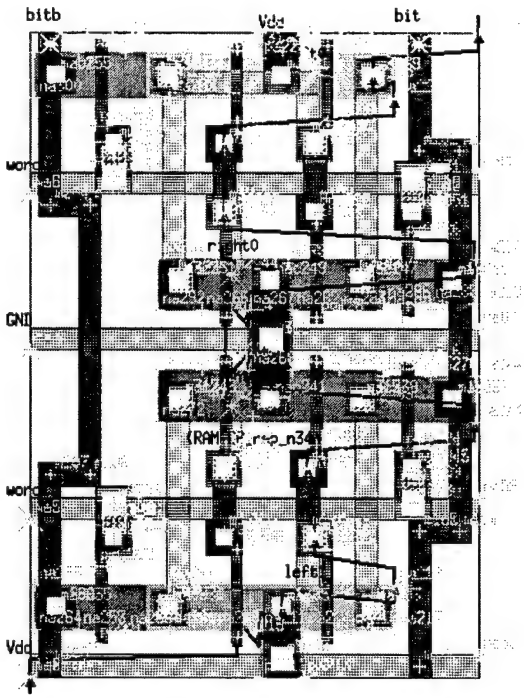


Fig. 4.2 MasterPort's area-critical path analysis of a 6T cell pair.

n	Design rule	O_{nx}	O_{ny}	O_n	Normalized Score
1	**Ohmic width**	**648**	**263**	**911**	**19.5**
2	Gate metal to ohmic spacing	792	275	1067	14.8
3	**Active overlap of contact**	**788**	**526**	**1314**	**8.4**
4	**Contact width**	**403**	**539**	**942**	**8.1**
5	**Minimum transistor width**	**0**	**263**	**263**	**5.6**
6	Gate metal spacing	2	268	270	4.3
7	Gate metal overlap of contact	12	552	564	3.6
8	Metal 1 spacing	257	3	260	2.8
9	Metal 1 width	256	0	256	2.7
10	Minimum transistor length	382	0	382	2.1
11	Gate metal overlap of active	3	265	268	1.6
12	Metal 3 spacing	72	0	72	1.0

Table 4.2 Ranked area-critical design rules.

values are not given in the table to maintain their confidentiality. As indicated in the table, the ohmic width rule is the most area-critical design rule in the 2 k-byte RAM. This design rule, however, is dependent on three other design rules: active overlap of contact, contact width, and minimum transistor width. In other words, the ohmic width design rule cannot be reduced (and produce an improvement in layout area) unless the other design rules upon which it is dependent are also proportionally reduced. The interdependent design rules are shaded in the table and are hereafter referred to as the "source/drain area" design rules.

4.2 CGaAs Design Rule Scaling Costs

In the previous section, the area-critical design rules were identified and ranked according to their ability to impact layout area. The dependencies between these design rules have also been determined, which facilitated the formation of interdependent area-critical design rule sets. To perform a cost/benefit analysis of these area-critical design rules, the impact of scaling these rules on die cost must be determined.

Scaling research and development, along with the depreciation of newly acquired capital equipment, can be amortized over the number of wafers that are fabricated in the scaled process. Since CGaAs is a low volume process, scaling research, development, and capital equipment depreciation significantly impact overall wafer cost. For this reason, a cost benefit analysis of the CGaAs design rules is especially appropriate.

4.2.1 Scaling Cost Estimates

Motorola process engineers have provided the estimated research, development, and capital equipment acquisition costs associated with scaling the CGaAs design rules by 10, 20, 30, and 40%. These cost estimates, given in Table 4.3, are normalized to one to maintain the confidentiality of the actual figures.

The cost estimates of Table 4.3 indicate two important points. First, the cost of linearly scaling the CGaAs design rules is not evenly distributed among the design rules. At different scaling factors, different rules become more critical and/or expensive to scale. Second, the costs associated with scaling a design rule increases at non-uniform rates. For example, it costs the same to reduce the via width design rule by 40% as it does to reduce it by 30%. This is true because a 30% reduction of the via width requires a transition to tungsten plug technology, which can be used up to a 40% width reduction without

CGaAs design rule	**Normalized R&D + Depreciation**			
	10%	**20%**	**30%**	**40%**
Minimum gate width	1	1	2	3
Minimum gate length	1	2	4	10
Ohmic area	1	1	2	4
FET spacing	2	4	8	12
Gate metal spacing	1	1	4	10
Gate metal to ohmic spacing	2	4	6	10
Via 0, 1, & 2 width	2	3	8	8
Metal 1, 2, & 3 width and spacing	1	2	3	10

Table 4.3 Normalized CGaAs design rules scaling cost estimates.

additional cost. Another example of this phenomenon occurs when scaling the metal spacing rule. As indicated in Table 4.3, there is a large increase in scaling cost for this design rule between the 30 and 40% scale factors. This occurs because a chemical-mechanical polishing (CMP) planarization technique is required to achieve a 40% reduction in interconnect spacing without incurring significant yield losses. However, with a 30% or smaller reduction of this rule, CMP planarization is not required.

The next step in calculating scaling cost estimates for use in cost/benefit analysis involves forming sets of disjoint design rules. Disjoint design rules can be scaled independently of each other and have non-overlapping scaling costs. The source/drain area design rules are a set of interdependent design rules that form a single set of design rules which should be simultaneously and proportionally scaled. Since the metal/via width and spacing design rules have scaling research, development, and capital equipment requirements that are a subset of the source/drain area design rule scaling requirements, the metal/via width and spacing design rules should be included in the source/drain interdependent design rule set. Table 4.4 describes four disjoint CGaAs design rule sets. The research, development, and capital equipment acquisition expense estimates for scaling these disjoint sets by 10, 20, 30, and 40% were obtained from Motorola process engineers and are normalized to one to maintain the confidentiality of the actual cost estimates. The actual design rules that are included in each disjoint set of Table 4.4 is given in Table 4.5.

CGaAs design rule	**Normalized R&D + Depreciation**			
	10%	**20%**	**30%**	**40%**
Source/drain area, via/metal width & spacing	5	7	15	25
Gate metal to ohmic spacing	2	4	6	10
Gate metal spacing	1	1	4	10
Miscellaneous spacing	2	4	8	12
Total (linear shrink):	**10**	**16**	**33**	**57**

Table 4.4 Disjoint CGaAs design rule sets and scaling costs.

Disjoint design rule set	CGaAs design rule
Source/drain area, via/metal width & spacing	Ohmic width Active overlap of contact Contact width Contact spacing Minimum transistor width Via 1 and 2 width Via 1 and 2 spacing Metal 1, 2, 3, overlap of contact/via Metal 1, 2, 3 width Metal 1, 2, 3 spacing
Gate metal to ohmic spacing	Gate metal to ohmic spacing
Gate metal spacing	Gate metal spacing
Miscellaneous spacing	Active spacing Gate metal overlap of contact Gate metal extension beyond active

Table 4.5 Disjoint CGaAs design rule composition.

The cost of performing a 10, 20, 30, and 40% linear shrink of all design rules is the sum of all disjoint set costs, and is given in Table 4.4. Linear scaling costs do not include the costs associated with scaling the transistor gate length because 0.5 μm CGaAs gate length reduction efforts have already been completed.

4.2.2 Disjoint Area-Critical Design Rule Set Formation

The computation time required for cost/benefit analysis can be reduced by considering only those design rules that are most likely to provide significant improvements in area and performance. A method for producing these design rule sets was explained and justified in section 2.3.2. In section 4.1, the CGaAs area-critical design rules were identified and ranked according to their ability to impact layout area. By examining the highest ranked area-critical design rules of Table 4.2, and by considering the interdependencies between them, three sets of area-critical design rules that are likely to provide a significant reduction in area with minimal cost can be formed and are given in Table 4.6.

The design rule scaling cost data in Table 4.4 can be used to estimate the costs of reducing the three area-critical design rule sets of Table 4.4 by 10, 20, 30, and 40%. These

CGaAs design rules	Area-critical design rule sets		
	Set 1	Set 2	Set 3
Source/drain area, via/metal width & spacing	X	X	X
Gate metal to ohmic spacing		X	X
Gate metal spacing			X

Table 4.6 Area-critical design rule sets.

cost estimates, given in Fig. 4.3, are compared with the cost of performing a linear shrink of the same proportions.

The costs represented in Fig. 4.3 are extremely large for scaling proportions beyond 30%. This reflects the costs that must be incurred to acquire manufacturing equipment with advanced capabilities, as well as the additional research and development required to aggressively scale the design rules. It is reasonable, from both a process engineering and a financial viability standpoint to assume an upper bound on scaling costs. For purposes of this analysis, a $25 (normalized) spending cap is assumed. This represents the cost of performing a 25% linear shrink of the design rules (obtained through extrapolation), which is a typical scaling factor for next generation processes.

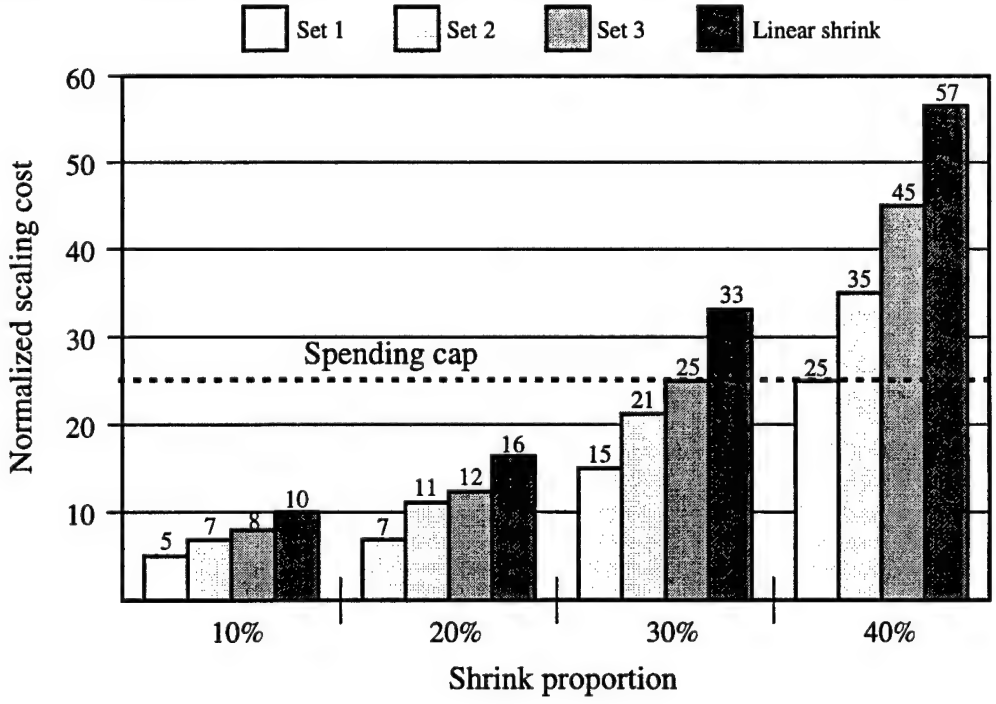


Fig. 4.3 Disjoint area-critical design rule set scaling costs.

As shown in Fig. 4.3, a 30 and 40% linear shrink, and a 40% scaling of the second and third disjoint area-critical design rule sets exceeds the $25 spending limit. Design points that correspond to these cost-infeasible points will be indicated throughout this analysis.

Another important component of wafer cost is the starting material and processing costs. CGaAs starting material costs significantly more than CMOS wafers. The cost of processing 4- and 6-inch GaAs wafers, including labor, occupancy, and chemicals, has been presented by Vitesse Semiconductor Corporation [156], and is assumed to be similar to CGaAs wafer processing costs.

4.3 Area and Performance Impact Analysis

The next step in design rule cost/benefit analysis is to determine how reducing each disjoint area-critical design rule set impacts overall performance and area. Until now, no tool has been available for conducting the type of benefit analysis necessary for design rule cost/benefit analysis. For this task, the PUMA RAM compiler is used to create near-optimal power-delay curves for a range of scaled processes from which power and delay improvement data can be extracted. The PUMA RAM compiler is also used to obtain area improvement data by generating RAM layout in each scaled process.

To illustrate this process, Fig. 4.4 gives the near-optimal power-delay curves for RAMs generated in five different CGaAs processes. The first process is the baseline, or unmodified 0.5 μm process. The other four processes have their set 1 design rules (source/drain area and metal/via pitch) reduced by 10, 20, 30, and 40%. The average vertical distance between the baseline and scaled process near-optimal power-delay curves represents the average power improvement ($\Delta power_{ave}$) that can be achieved in the scaled process. This average vertical distance (in mW) is computed between upper and lower delay bounds (d_{max} and d_{min} respectively). The region between these upper and lower limits is the range of delay values for which the RAM circuit is both feasible and practical. Fig. 4.4 shows the placement of the d_{max} and d_{min} boundaries within the RAM power-delay design space.

The average delay improvement ($\Delta delay_{ave}$) is found by calculating the average horizontal distance (in nanoseconds) between near-optimal power-delay curves. These

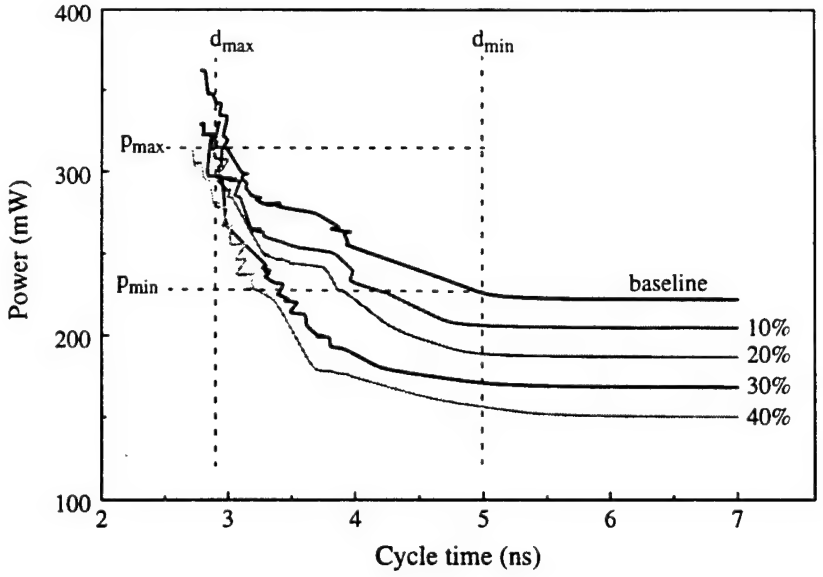


Fig. 4.4 2 k-byte RAM near-optimal power-delay curves.

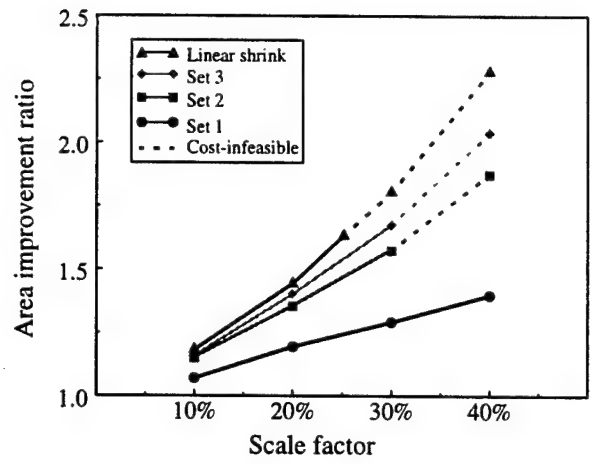
calculations are made for the region between an upper and lower bound on power dissipation (p_{max} and p_{min} respectively) that represent a region of available data for comparison, e.g., p_{min} is determined as the baseline RAM power dissipation at d_{max}. The average area improvement ($\Delta area_{ave}$) is found by generating and comparing RAMs for various points along the near-optimal curves within the power and delay boundaries.

This type of analysis was conducted for all three disjoint area-critical design rule sets of Table 4.4 and for a linear shrink of all design rules except minimum gate length. Fig. 4.5 illustrates the average area, power, and delay improvement ratios, defined in equations 4.1, 4.2, and 4.3 respectively, that each design rule set provides when reduced through a range of shrink proportions. In equations 4.1 through 4.3, A_b, P_b, and D_b represent the average baseline area, power, and delay values within the bounded power and delay design space. The results of a linear shrink analysis are also shown in Fig. 4.5 for comparison purposes.

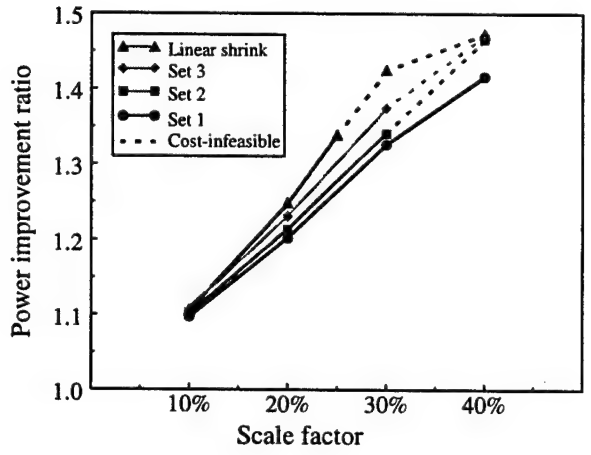
$$\text{Area improvement ratio} = \frac{A_b}{A_b - \Delta area_{ave}} \qquad (4.1)$$

$$\text{Power improvement ratio} = \frac{P_b}{P_b - \Delta power_{ave}} \qquad (4.2)$$

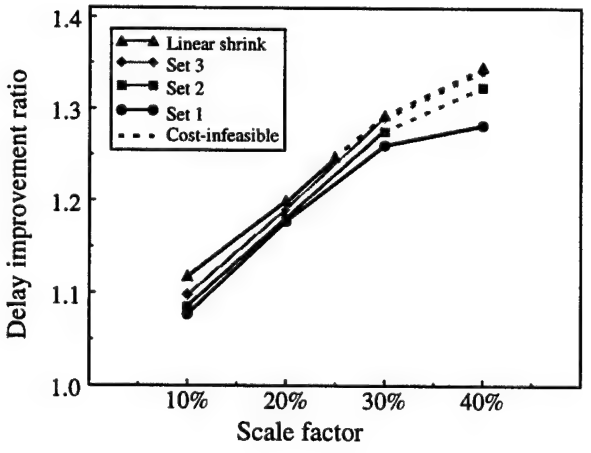
$$\text{Delay improvement ratio} = \frac{D_b}{D_b - \Delta delay_{ave}} \qquad (4.3)$$



(a) area



(b) power



(c) delay

Fig. 4.5 Area, power, and delay impact analysis results.

Nonlinear design rule scaling allows certain area-critical geometries to be scaled beyond what would normally be infeasible due to cost limitations. The plots in Fig. 4.5 indicate the area, power, and delay advantages of doing so. Fig. 4.5(a) shows that scaling the third area-critical design rule set by 30%, which is cost-feasible, produces a greater improvement in area than a 25% linear shrink of all design rules, which is also cost-feasible. Fig. 4.5(b) indicates a that by either scaling the third area-critical design rule set by 30%, or by scaling the first area critical design rule set by 40%, a greater improvement in power is produced than a 25% linear shrink. Fig. 4.5(c), shows that four cost-feasible scaling points, including a 30% reduction of sets 1, 2, or 3, or a 40% reduction of set 1, produce greater improvements in delay than a 25% linear shrink. Therefore, nonlinear design rule scaling can provide cost-feasible area and performance benefits that exceed those obtained through linear scaling. This is accomplished by aggressively scaling the most area-critical design rules and then making up for the extra cost by not scaling other design rules as aggressively, or not at all.

4.4 Cost/Benefit Plot Formation and Interpretation

The area, power, and delay improvement data from the area and performance impact analysis can be combined with die cost estimates to form a cost/benefit ratio for a range of shrink proportions. The formula for calculating the cost/benefit ratio, which was originally given in chapter 2, is repeated here:

$$\text{Cost/benefit ratio} = \frac{\text{Die cost}}{\frac{P_b}{P_b - \Delta\text{power}} \cdot \frac{D_b}{D_b - \Delta\text{delay}}} \qquad (4.4)$$

The product of the power and delay improvement ratios form the denominator of the cost benefit ratio and can be referred to as the overall benefit ratio. Fig. 4.6 illustrates the benefit ratios for 10, 20, 30, and 40% reductions of the area-critical design rule sets of Table 4.4, and compares them to the benefit ratios of a linear shrink.

Fig. 4.6 confirms the results mentioned previously and shown in Fig. 4.5(a), (b), and (c). There are three cost-feasible points that provide a higher overall performance benefit than a 25% linear shrink. These points include a 30 and 40% reduction of the first

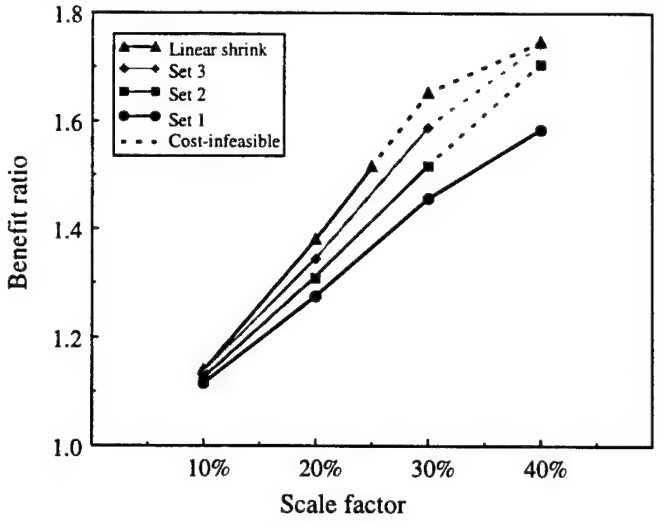


Fig. 4.6 Area-critical design rule benefit ratios.

area-critical design rule set and a 30% reduction of the second area-critical design rule set. Moreover, reducing the first area-critical design rule set by either 30 or 40% will provide this increased overall performance advantage at a lower scaling cost than a 25% linear shrink.

When calculating CGaAs die costs, a representative baseline die area is assumed, which is A_b multiplied by a constant c to produce a typical die size for the manufacturing facility. This baseline die area is modified by $\Delta area_{ave}$ according to the following equation:

$$\text{Die area} = c \cdot (A_b - \Delta area_{ave}) \qquad (4.5)$$

This modified die area is used for calculating both the die yield and the number of dies per wafer. Using this technique, the area improvement data of Fig. 4.5(a) was used to create die cost estimates for three different CGaAs manufacturing scenarios having low, moderate, and high wafer counts. Scaling costs for different wafer volumes should include appropriate adjustments for increased processing capacity and cost of capital. Typical cost of capital adjustments in the semiconductor industry range from 10 to 15% annually [157]. Fig. 4.7 illustrates how scaling the area-critical design rules, as well as performing a linear shrink, affects die cost.

The plots of Fig. 4.7 show that additional scaling efforts and expenses become justified on a die cost basis alone as wafer counts increase. Fig. 4.7(a) and (b) indicate that

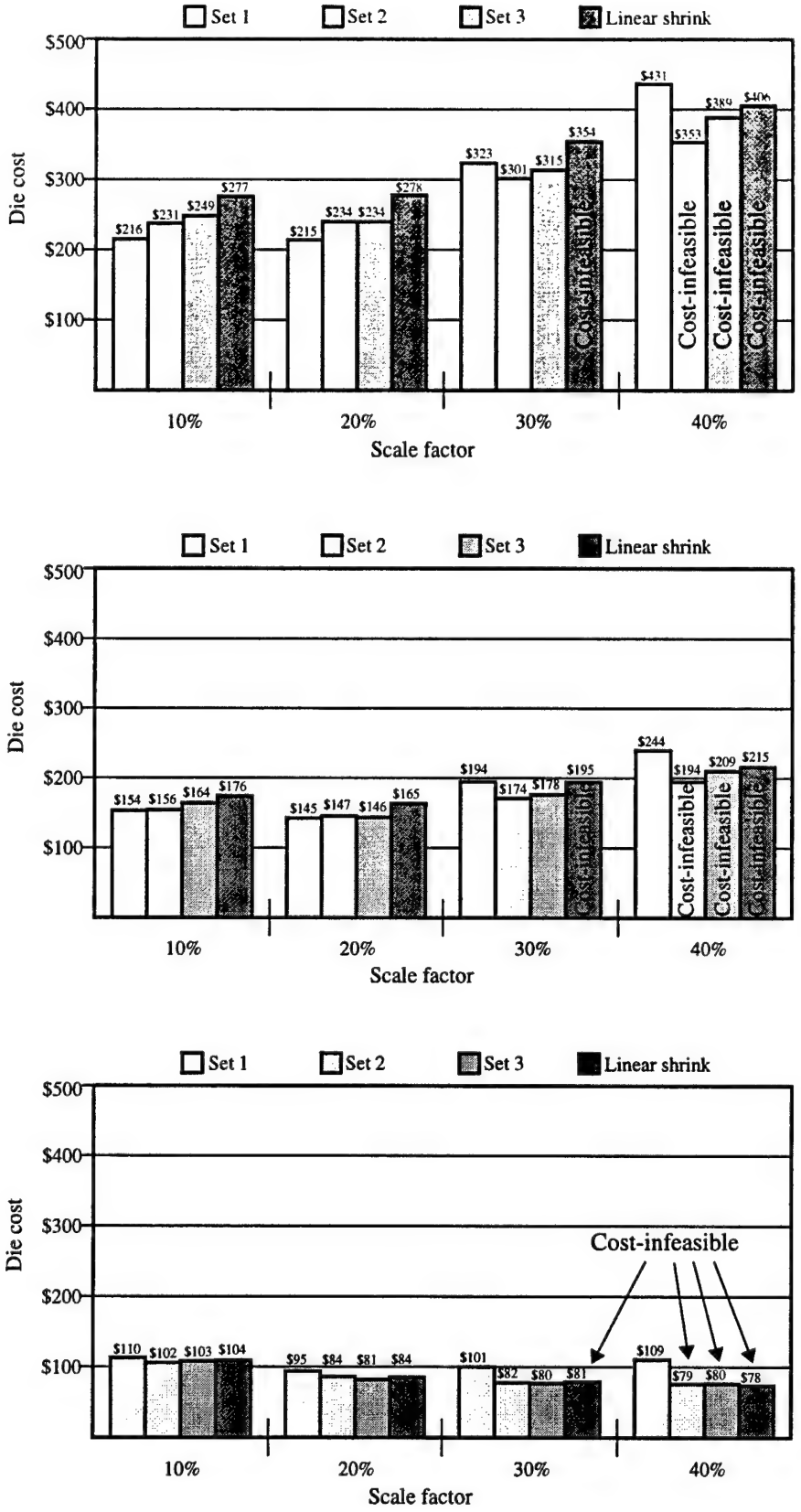
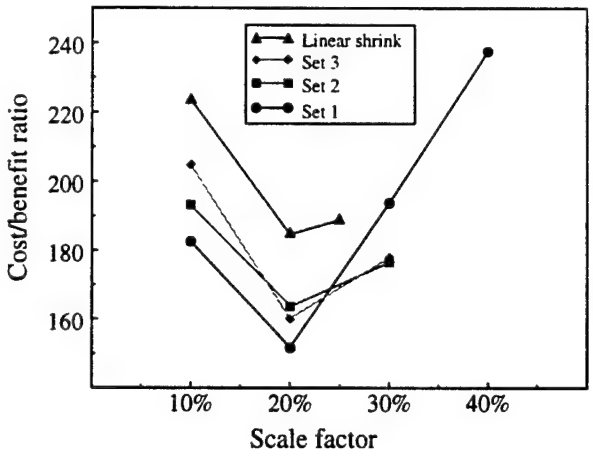


Fig. 4.7 CGaAs die cost estimates.

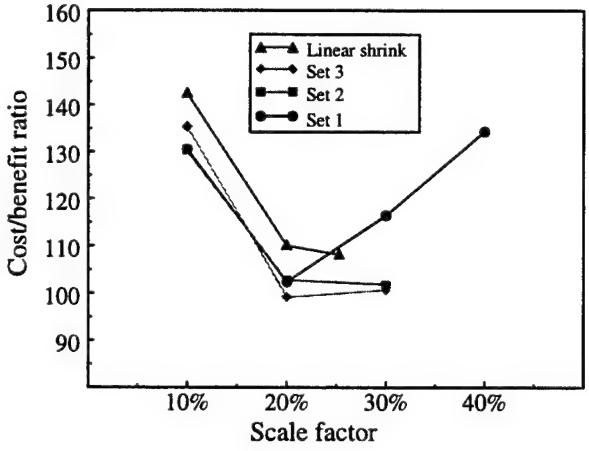
for low and moderate wafer counts, the lowest die cost is achieved when the first area-critical design rule set is scaled by 20%. For a high wafer count, a 40% linear shrink produces the lowest die cost, as shown in Fig. 4.7(c). When the cost data of Fig. 4.7 is combined with the benefit data of Fig. 4.5(b) and (c) according to equation 4.4, the cost/benefit ratio plots of Fig. 4.8 result.

Cost/benefit plots are organized to guide the process engineer in selecting cost efficient reduction ratios for the horizontal design rules under consideration. Each curve in the figures corresponds to an area-critical design rule set and represents the resulting cost/benefit ratio as that design rule set is scaled through a range of proportions. Only cost-feasible scaling factors are considered for each of the area-critical design rule sets. For each curve in Fig. 4.8(a), the minimum cost/benefit ratio occurs at the 20% scale factor. Even though performance and area improvements can result from shrinking these design rule sets beyond 20%, the costs associated with achieving these improvements are high. A 20% reduction is the point of diminishing returns from a cost/benefit perspective; this suggests that each design rule set should be scaled by only 20% before reconsidering all other design rules for reduction.

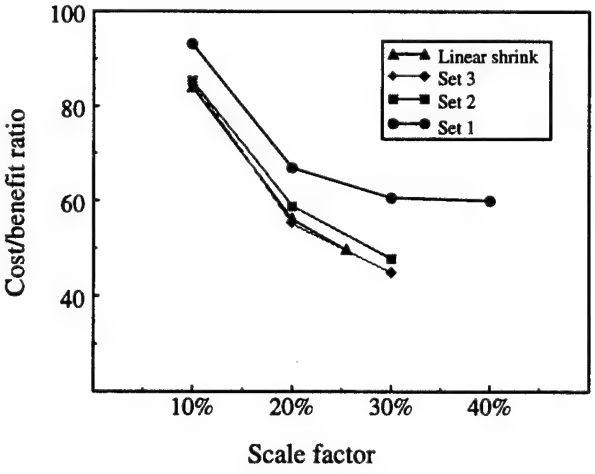
The curves of Fig. 4.8(a) also illustrate the differences in overall cost effectiveness (for a low wafer count scenario) between the three disjoint area-critical design rule sets of Table 4.4. The minimum overall cost/benefit ratio occurs when the first design rule set (source/drain area and via/metal pitch design rules) is scaled by 20%. This suggests that the nonlinear shrink of the CGaAs design rules (for low wafer count scenarios) should begin with a 20% reduction of these particular design rules. Moreover, the power, delay, and area improvement that can be achieved by scaling these design rules by 20% forms the baseline process for the next analysis iteration. This new baseline process is likely to have a different set of area-critical design rules than the first iteration, and will therefore produce different area-critical design rule sets to be analyzed for area and performance improvement. Before proceeding with another iteration, however, the upper bound on cost must be lowered by \$7 (normalized), which is the cost of scaling the first area-critical design rule set by 20%. By continuing this iterative procedure, a global minimum for the cost/benefit ratio will eventually be found as the associated scaling costs continue to rise with each iteration.



(a) low wafer count



(b) moderate wafer count



(c) high wafer count

Fig. 4.8 Area-critical design rule cost/benefit plots.

For low wafer volumes, scaling costs strongly influence wafer costs because they are amortized over a smaller number of wafers. In this scenario, the minimum cost/benefit ratio tends to occur at reduction proportions where the scaling costs are low. For larger wafer counts, the scaling costs are amortized over a large number of wafers and therefore have a smaller impact on wafer cost. Under these conditions, the cost/benefit ratio minimum tends to occur at reduction proportions that provide the best area and performance improvement, even if the associated scaling costs are high.

The cost/benefit curves of Fig. 4.8(b) and (c) illustrate the impact of higher wafer counts on cost and performance. As the wafer count increases, the cost/benefit curves slope downward to reflect the amortization of the scaling costs. Fig. 4.8(b) shows that for moderate wafer volumes, the minimum cost/benefit ratio occurs when the third area-critical design rule set is scaled by 20%. This reflects a compromise between scaling costs, area reduction, and performance improvement. Fig. 4.8(c) shows that for large wafer counts, the minimum cost/benefit ratio occurs when the third design rule set is scaled by 30%. At this point, the greatest amount of area and performance improvement is achieved for all cost-feasible scaling points, including the maximum feasible linear shrink.

4.5 Threshold Voltage Cost/Benefit Analysis

The previous sections have demonstrated how a cost/benefit analysis for process design rules is conducted. The results of this analysis can be used by the process engineer in selecting design rule scaling ratios that make an effective tradeoff between die cost and performance. Cost/benefit analysis, however, can also be used to analyze the cost performance tradeoffs associated with scaling transistor operating parameters, such as threshold voltage.

Along with providing design rule scaling cost estimates, Motorola process engineers have also provided research, development, and capital equipment cost estimates for reducing the CGaAs transistor threshold voltage by 10, 20, 30, and 40%. Unfortunately, reducing the threshold voltage does not reduce die area, which makes it impossible to recover some of these expenses through higher yields or by having more dies per wafer. Table 4.7 gives the normalized research, development, and capital

Reduction proportion	Normalized Cost
10%	5
20%	10
30%	15
40%	18

Table 4.7 CGaAs transistor threshold voltage reduction cost estimates.

equipment acquisition cost estimates for reducing the CGaAs transistor threshold voltage by a range of practical reduction proportions.

The PUMA RAM compiler was used to evaluate the performance impact of scaling the threshold voltages by as much as 40%. Fig. 4.9 shows the 2 k-byte RAM near-optimal power-delay curves for five different process. The first process is the baseline, or unmodified 0.5 μm CGaAs process. The next four processes have identical horizontal design rules, but differ by having threshold voltages that are reduced by 10, 20, 30, and 40%.

As indicated by the near-optimal power-delay curves, reducing the threshold voltage shifts the curves upward and to the left in the power-delay design space. As the curves move upward, average RAM power dissipation increases. As the curves move to

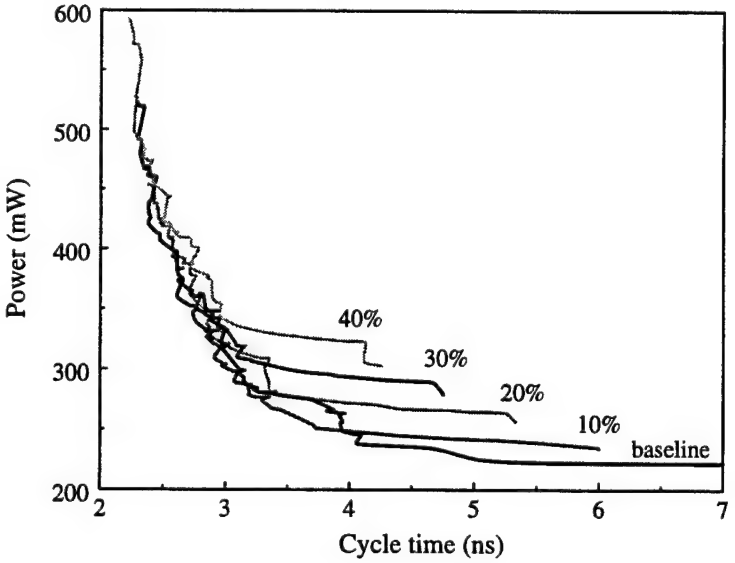


Fig. 4.9 RAM power-delay curves for threshold voltage evaluation.

the left, average delay decreases. When compared to the baseline process, a threshold voltage-shifted process increases power dissipation for lower-clock rate RAMs and a decreases power dissipation for higher clock rate RAMs. Of course, the threshold voltage-shifted processes are capable of producing faster RAMs than the baseline process, and these RAMs dissipate more power at their maximum operating frequencies.

Fig. 4.10 illustrates the power and delay improvement ratios that result from reducing the CGaAs transistor threshold voltage by 10, 20, 30, and 40%. The rising delay improvement curve and falling power improvement curve result from the near-optimal power-delay curves moving upward and to the left in the power-delay design space. As indicated in the figure, the resulting benefit curve, which is the product of the power and delay improvement ratios, is nearly flat over the scale factors under consideration.

Fig. 4.11 shows the cost-performance tradeoffs being made when scaling the threshold voltage using both a cost/benefit and a cost/delay ratio. These plots have been created for low, moderate, and high wafer count scenarios.

For each wafer count scenario, the cost/benefit and cost/delay plots slope upward with larger threshold scale factors. This indicates that the rate at which die cost increases due to threshold voltage scaling exceeds the rate at which the resulting performance and/or delay improves. Therefore, CGaAs transistor threshold voltage scaling immediately produces diminishing returns from an SRAM cost/benefit and cost/delay perspective. This does not mean that reducing the threshold voltage is a mistake, or that it would not

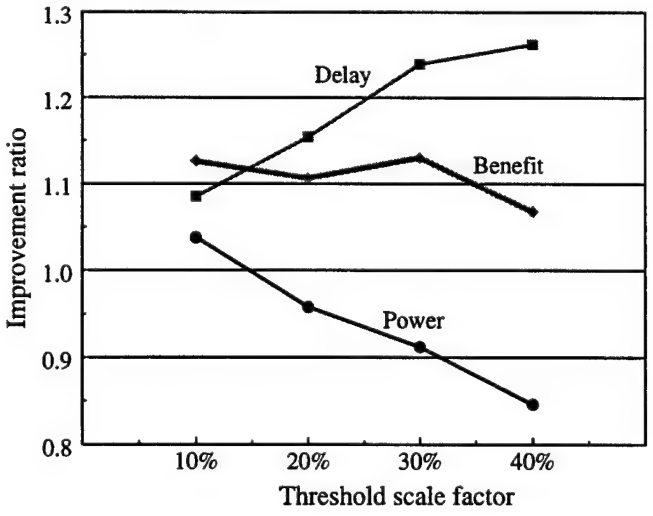
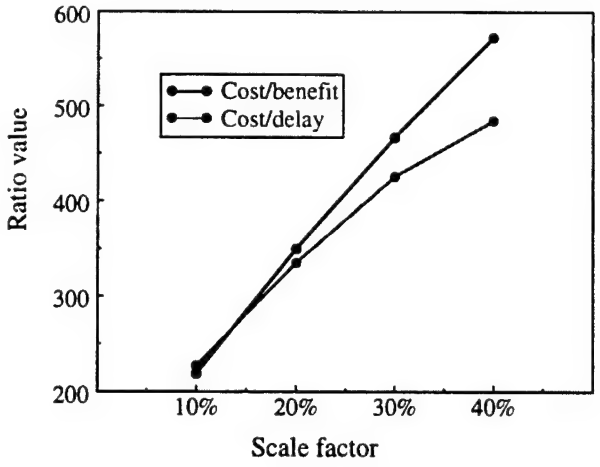
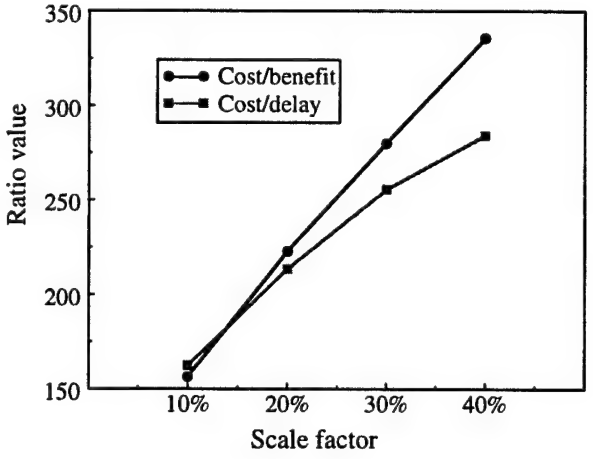


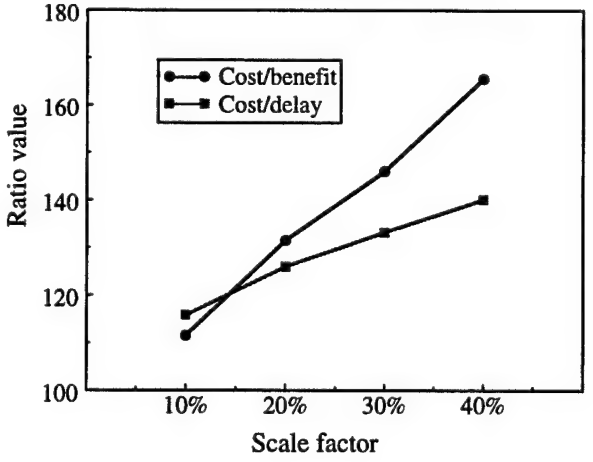
Fig. 4.10 Threshold voltage scaling power, delay, and benefit ratios.



(a) low wafer count



(b) moderate wafer count



(c) high wafer count

Fig. 4.11 Cost/benefit and cost/delay plots.

lead to an overall increase in product competitiveness and profitability. Reducing the transistor threshold voltage certainly provides a significant increase in performance. What this analysis does suggest is that this delay improvement comes at a high price, both in terms of scaling costs and power dissipation, and that, from an SRAM perspective, this cost can only be justified by a requirement for delay improvement beyond what horizontal design rule scaling can provide.

From a logic or datapath perspective, an entirely different conclusion may be reached. As the transistor threshold voltage is reduced, logic transistor widths may also be decreased, which will result in an improvement in layout area. This will reduce die cost, and may make threshold voltage reduction a cost effective alternative for reducing die area and improving performance.

4.6 Conclusion

Motorola's 0.5 μm CGaAs process is an excellent example of a process that has undergone the first phase of nonlinear design rule scaling, transistor gate length reduction, and is awaiting the second phase, a cost/benefit analysis of the horizontal design rules. Design rule cost benefit/analysis is an iterative procedure for determining the design rule shrink proportions that reach an effective compromise between cost, performance, and area. A single iteration of this analysis procedure was conducted for the CGaAs design rules using a 2 k-byte SRAM with a square aspect ratio as the representative design. MasterPort's area-critical path analyzer was used to determine which leaf cells and design rules determined the RAM layout x- and y-dimensional area-critical paths. These design rules were then quantified and ranked according to their ability to impact minimum layout area. The most area-critical design rule was found to be the minimum ohmic area rule, which is dependent upon the ohmic contact width, active overlap of contact, and minimum transistor width design rules. These interdependent design rules formed part of the first of three area-critical design rule sets.

Research, development, and capital equipment acquisition cost estimates for scaling the CGaAs design rules through a range of practical reduction ratios were made available from Motorola process engineers. Along with these estimates came an understanding of the degree of development overlap between design rules. This

information facilitated the formation of disjoint design rule sets, which are sets of design rules that can be scaled independently of each other and do not have overlapping scaling costs. The three most area-critical disjoint design rule sets were then analyzed for area and performance impact.

The PUMA RAM compiler was used as the instrument for determining the effects of scaling each area-critical design rule set on performance and area. Area and performance improvement data were combined with cost estimates to produce a cost/ benefit plot for each of the three disjoint area-critical design rule sets scaled by as much as 40%. These plots can be used by the process engineer to guide the selection of design rule reduction proportions that make an effective tradeoff between cost and performance.

A cost/benefit analysis of CGaAs transistor threshold voltage reduction was also conducted. A performance impact analysis, conducted with the PUMA RAM compiler, found that decreasing the threshold voltage shifts the RAM near-optimal power-delay curve upward and to the left in the power-delay design space. This results in an increase in power dissipation for lower frequency RAMs, a decrease in power dissipation for higher frequency RAMs, and shorter delays. This cost/benefit analysis also found that the costs associated with reducing transistor threshold voltages increase faster than the resulting improvement in performance. This suggests that, from an SRAM perspective, threshold voltage reduction can only be justified by a requirement for increased performance beyond what horizontal design rule scaling can provide.

CHAPTER 5
CONCLUSION

The scaling of the integrated circuit manufacturing processes has been the primary driver behind the tremendous increase in microprocessor performance witnessed in recent years. To simplify the porting of processor designs from one process generation to the next, IC process engineers have been performing linear shrinks of the horizontal design rules. While this scaling technique eases the task of design migration, it can be very costly from a research, development, and capital equipment perspective. To this day, most major semiconductor manufacturers are still performing linear or hybrid shrinks of their major CMOS processes. These linear shrinks are performed "blindly," without considering the cost, performance, or area impact associated with reducing the individual design rules.

Past layout migration tools have not been effective for porting and re-optimizing entire VLSI designs from one IC process to the next. This lack of CAD capability has been the primary motivation for linearly shrinking the process design rules. However, layout migration CAD tools have been steadily improving and have begun to reach the point of acceptability in solving practical problems. This trend will surely continue; as it does, nonlinear design rule scaling will become more feasible.

The improvement of layout migration CAD tools is only one factor motivating the use of nonlinear design rule scaling — managing the rapidly increasing costs associated with shrinking the physical dimensions of transistor components is another. The cost of performing a linear shrink is not evenly distributed among the design rules. Instead, much of the scaling cost centers around a few stubborn design rules that can be very difficult, and hence expensive to scale. By adopting a nonlinear scaling approach, it is no longer necessary to commit large amounts of resources to the reduction of a few stubborn design rules. This will reduce design rule scaling costs and provide faster availability of advanced processes.

The only remaining question is, when will nonlinear design rule scaling become a necessity, instead of being one of several scaling options. According to a major semiconductor manufacturer [158], full nonlinear scaling will be inevitable once minimum feature sizes fall below 0.1 μm. At this point, certain layers will become very difficult and expensive to shrink. When nonlinear design rule scaling is adopted, a quantitative methodology for determining the most cost effective scale factors for the various design rules will be essential. This thesis has made several contributions in the area of nonlinear design rule scaling, including both analysis methodologies and CAD tools.

5.1 Contributions

This thesis introduces a methodology for determining scaled horizontal process design rule values that reach an effective tradeoff between not only cost and area, but performance. This is accomplished with a procedure that iteratively finds the design rules that have the greatest impact on minimum layout area, and reduces them to their points of diminishing return from a cost, area, and performance perspective. The primary instrument for performing this analysis is a process-independent RAM compiler.

The PUMA RAM compiler is a unique tool that was developed to fulfill two major roles. First, to provide comprehensive performance and area comparisons between IC processes having different design rules, and second, to generate optimized embedded SRAMs for the PUMA PowerPC microprocessor. This tool is the first truly process-independent RAM compiler that is capable of both generating and optimizing RAM layout for any IC process. Although the layout generator was tuned to produce CGaAs layout, it is capable of generating RAMs in any complementary technology. The PUMA RAM compiler not only generates optimized RAM layout, but also produces near-optimal power-delay curves for a given process. This curve, which indicates the range of RAM power and delay values that can be achieved in a particular process, is used for making fair and comprehensive comparisons between different processes. This thesis also describes optimization algorithms for exploring the large SRAM transistor size design space, and gives an innovative approach for optimizing an entire synchronous SRAM.

Most commercial SRAM designs employ a memory bit cell that has been designed and verified by the foundry and is not meant to be altered in any way by the RAM designer. One of the most unique and powerful capabilities of the PUMA RAM compiler is its ability to adjust the sizes of the transistors in the memory bit cell. During the optimization process, the compiler allows the transistors within the memory bit cell to grow in order to supply additional current to the sense amplifiers, or to help speed the data write time. Each new memory bit cell considered is verified for stability to ensure correct operation of all designs on the near-optimal power delay curve.

The PUMA RAM compiler has produced two SRAM designs for Motorola's 0.5 μm CGaAs process. The first is a stand-alone 2 k-byte test SRAM and the second is an embedded 1 k-byte instruction cache for the PUMA PowerPC microprocessor. These designs have returned from fabrication and will soon be tested.

This thesis gives the results of a design rule cost/benefit analysis for Motorola's 0.5 μm CGaAs process. The first iteration of this analysis determined that the transistor source/drain area design rules were the most area-critical, and that an effective tradeoff between cost, area, and performance could be reached by reducing these, and other area-critical design rules according to the number of wafers over which the scaling costs are amortized. Not only does this analysis determine the way to make an effective tradeoff, but they show that significant improvements in performance and area can be achieved with the reduction of just a few design rules. Since the CGaAs design rules are very coarse for a 0.5 μm technology, there is opportunity for much improvement.

Finally, a cost/benefit analysis of CGaAs transistor threshold voltage scaling is described in this thesis. Through PUMA RAM compiler-based performance evaluations and die cost estimations, it was shown that CGaAs transistor threshold voltage scaling is expensive (with respect to expected benefits), compared to horizontal design rule scaling.

5.2 Future Work

This thesis is the first to present a quantitative method for cost effective nonlinear design rule scaling. This work points to several related topics that could produce interesting and valuable results. Much of the present work concentrates on analyzing area and performance improvement from an embedded CGaAs RAM perspective. These

results become more applicable as microprocessors utilize a larger percentage of their transistors as embedded RAM. However, it would also be interesting to explore performance and area improvement from both a CMOS and a random logic perspective.

The quantitative approach for exploring the design rule reduction space given in this thesis is applicable to random logic. However, a different tool is required for performing the optimization of random logic blocks. For a given placement of standard cells, this optimization tool must be capable of generating, compacting, and optimizing layout for any target process. Such a tool would also be required to route the design in an optimal manner using any number of routing layers, specified at run time. Finally, the tool's capabilities must include transistor and buffer size optimization. Such a tool would allow a comparison to be made between the benefits of adding additional metal layers, or scaling the existing metal layers.

A move to nonlinear design rule scaling is inevitable. It is crucial that comprehensive, quantitative methodologies to support it be in place when this occurs. I hope that this research will motivate others to develop even more comprehensive solutions to nonlinear design rule scaling.

APPENDICES

APPENDIX A

MASTERPORT

MasterPort [150] is a collection of cell library generating tools that can be accessed from a menu-driven graphical user interface. Before these tools can be used to generate a cell for a particular technology and process, generic layout geometry must be provided. This geometry can be imported from a GDS2 file, or created from scratch using MasterPort's geometry editing tools. The MasterPort database represents this geometry using landmarks and objects.

A1. Geometry Representation

Landmarks are points that mark the end points, centers or edges of geometric objects. Each landmark has an associated x and y coordinate, a text label used for unique identification, and a node label to indicate electrical connectivity. Four types of objects can be represented: *boxes*, *strips*, *polygons*, and *squares*. Fig. A1 illustrates the four types of representable objects with their associated landmarks. Each object has a mask layer, such as "polysilicon" or "active", and if applicable, a width value associated with it. Mask layers and widths are defined by the user and thus allow a wide variety of technologies to be correctly represented.

A2. Process Technology Representation

The design rules associated with a given IC technology are represented in the MasterPort database as symbolic *micros*. Each micro corresponds to a particular design

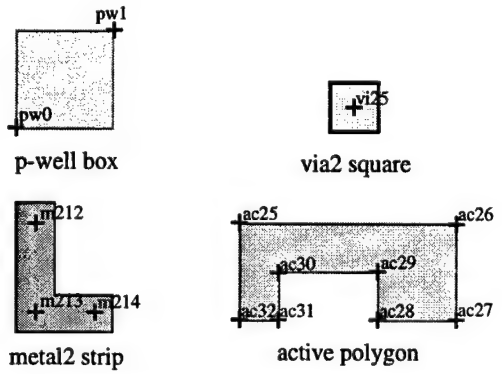


Fig. A1. Geometry objects and landmarks.

rule and has an associated value. For example, the polysilicon spacing rule is represented as "pys" in the MasterPort process technology database. The user sets the value of this and all other micros by creating a text file that indicates the values of each micro. A MasterPort utility is then used for converting and entering this text file into the process technology database. Micros have been established for governing design rules for and between the n-well, p-well, active, n+, p+, contact, poly1, poly2, metal1 through metal4, via1 through via3, bond opening, LDD, and ROM implants mask layers.

A3. Geometry Constraints

Once a generic cell geometry and a target process technology have been entered into the MasterPort database, the landmarks used to describe the generic geometry must be constrained. A constraint is a relationship between a reference landmark and a target landmark and has a direction, type and strength associated with it. The constraint direction is either *east* or *north* and indicates the location of the target landmark in relation to the reference landmark.

The constraint type indicates the nature of the constraint between the target and reference. There are four main types of constraints: *side, delta, align* and *corner.* Side constraints are used to indicate a minimum edge-to-edge distance between the geometries represented by the target and reference landmarks. Delta constraints allow the user to specify an arbitrary minimum side-to-side spacing between two landmarks and are used mainly for transistor sizing. Align constraints are used to ensure that two landmarks remain aligned. Corner constraints allow two geometries to move closer together if they are situated diagonally from each other and have room to move either above or beside each other. Corner constraints play a significant role in layout geometry compaction.

Each constraint has one of the following associated strengths: *free, attract,* or *attach.* A free constraint ensures that a minimum, DRC-correct spacing distance will be obtained without moving the target point toward the reference point. Attract constraints will attempt to move the target point toward the reference point as much as possible without moving other target points. This type of constraint is used to weakly enforce cell compaction. The attach constraint will move the target point as close as possible to the

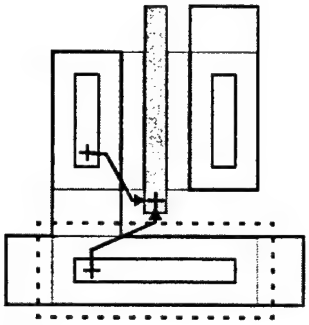


Fig. A.2. A pair of geometry constraints.

reference point without violating design rules and will move other target points in the process. This type of constraint is used to strongly enforce cell compaction.

Fig. A.2 illustrates a pair of simple geometry constraints. In this example, the polysilicon gate is constrained by the source/drain contact region with a side-east-attract constraint. This constraint is used to weakly enforce a minimum separation distance between the source/drain contact and the gate and would be symbolically evaluated to the "acgts" micro (active area contact to gate poly spacing). A second constraint is also shown in the figure between the source/drain well tie and the polysilicon gate. This constraint is a side-north-free constraint and serves to ensure a minimum separation distance between the well contact and the gate polysilicon. This constraint would be symbolically evaluated to the "wcpys" (well contact to poly spacing) micro.

Geometry constraints do not explicitly convey spacing information. Instead, they merely state relationships between points that are not to be violated in order to guarantee DRC and LVS correctness, and to ensure that the cell is compact. The process of converting these constraints into actual spacing distances is known as constraint evaluation.

The key benefit of using MasterPort versus a more traditional cell compaction methodology is that a single set of constraints can be generated that should be valid over a wide range of design rule values. These constraints are used to represent the designer's knowledge of how the cell should migrate from one process to another. MasterPort also allows a designer to edit these constraints to solve cosmetic issues and control where minimization and stretching will occur. Much of this work can be specified in the automatic constraint generation rules that MasterPort uses to initially constrain a cell. The

user can then edit these automatically generated constraints to make topological changes if a cell does not meet the user's requirements.

A4. Constraint Evaluation

The process of evaluating a cell's geometry constraints is an iterative procedure. Once all of the constraints have been correctly evaluated, a new cell geometry can be generated that is compatible with the specified target IC process. The evaluation procedure is as follows:

First, a check is made to ensure that no constraint cycles exist among the geometry landmarks. A constraint cycle is a set of two or more constraints that are interdependent, and cannot therefore be resolved. If a constraint cycle is found, the user is alerted and the evaluation procedure is halted. Once the constraints have been verified as non-cyclical, evaluation can be guaranteed to complete.

The next step in constraint evaluation involves making an initial pass through the constraints to produce a compacted cell. This process begins with converting each constraint into a sum of symbolic design rule micros that correctly represent the minimum spacing distance between the reference and target. For example, a side constraint between adjacent metal2 lines would be translated to the "m2s" micro, which represents the metal2 spacing design rule. When translation is complete, the process technology database is read to associate the actual value of a symbolic design rule to a specific distance.

Once the constraints have been converted to spacing distances, a check is made to eliminate multiple constraints on a single target that are in the same direction. In this case, the constraint with the largest evaluated distance is kept as the remaining constraint.

Attach constraints are then evaluated. This procedure may cause other target points to move as a result. It is then necessary to reevaluate the entire cell to ensure that no design rules were violated. This iterative process alternates between east and north constraints and continues until all attach constraints have been successfully evaluated without causing target points to shift. At this point, the attract and align constraints can be safely evaluated without requiring the iterative reevaluation of the entire cell. The final evaluated cell is then displayed in the MasterPort GUI for inspection and editing by the user.

A5. Constraint Editing

There are four main reasons why it would be necessary for the user to edit a cell's geometry constraints. If a constraint cycle is found to exist in the cell, the user must break the cycle by deleting or altering a constraint. If a DRC or LVS error is present in the evaluated cell, a constraint is either missing or in error. If the evaluated cell is not sufficiently compact, an unnecessary constraint may exist and must be deleted, or an existing constraint does not have the correct strength. Finally, certain cells may require a symmetrical layout to equally balance capacitive loads. If a symmetrical cell becomes unbalanced when evaluated, it is necessary to either edit or create additional constraints. The MasterPort GUI provides a constraint display and edit feature to ease the process of viewing and editing constraints.

A6. Hierarchical MasterPort

In order to generate a RAM cell library, MasterPort must understand the pitch-matching relationship between each of the RAM cells. To accomplish this, MasterPort is capable of evaluating a cell with two levels of hierarchy. The only restriction placed upon this hierarchy is that no geometry exist in the top level cell. All cell geometries must be placed in leaf cells that are individually and separately evaluated as previously described. If an imported cell has more than two levels of hierarchy, or has geometry in the top level cell, the cell is "smashed" and "tiled" down to two levels of hierarchy according to the user's specification.

A7. Hierarchical Constraint Generation

Prior to generating constraints on the top level cell, two types of landmarks are brought up from the leaf cells to the top level cell: port and abutment box locations. Ports indicate the location of nodes that electrically connect to adjacent cells. Abutment boxes mark the periphery of each cell and are used for determining the amount of overlap that should be used when placing adjacent cells. The landmarks associated with abutment boxes are then constrained in the top level cell with one of two types of constraints: *placement* and *pitch-matching*.

Placement constraints are of the same types, strengths, and directions as regular leaf cell constraints and are applied to abutment box landmarks. Placement constraints are

used to determine the location of leaf cells with respect to each other. Pitch-matching constraints, placed between *connection points* (landmarks associated with port locations), force the alignment of connecting ports on adjacent leaf cells. The pitch-matching constraint is different from a placement constraint in that it instructs the MasterPort constraint evaluation engine to insert delta constraints into the leaf cells that will guarantee the alignment of ports on adjacent leaf cells.

A8. Hierarchical Cell Evaluation

The evaluation of a hierarchical cell is accomplished by first evaluating each of the leaf cells in the same manner as previously described. Once the leaf cells are completely evaluated, the top level cell is evaluated. This may cause the insertion of delta constraints into the leaf cells (for pitch-matching), thus requiring the reevaluation of the leaf cells. The iterative procedure of evaluating the top level cell, then reevaluating the affected leaf cells continues until the top level cell successfully evaluates without requiring the reevaluation of a leaf cell. It is possible that this procedure could continue indefinitely. To eliminate the possibility of infinite loops in hierarchical evaluation, a user-defined upper limit is established for the number of evaluation iterations. Once this limit is reached, reevaluation of the offending leaf cells are terminated and the user is alerted that pitch-matching did not complete successfully.

Upon completion of the hierarchical evaluation procedure, the evaluated top cell is displayed in the MasterPort GUI for inspection and, if necessary, constraint editing. Evaluated top level and leaf cells can then be exported to GDSII, CIF, or the MasterPort Design Database Management System (DDMS) geometry format.

APPENDIX B

PUMA RAM COMPILER PROGRAMS

```
-------------- CGaAs SRAM Layout Generator ------------
usage: ram_layout -row <rows> -col <cols> -d1 <decode1> <name>
-row              Number of rows
-col              Number of columns
-d1               Primary column decoding ratio
[-d2 (0)]         Secondary column decoding ratio
[-x (8)]          Number of RAM cells between horizontal power rails
[-y (8)]          Number of RAM cells between vertical power rails
[-r (project.go)] Set process ruleset
[-p (project.go)] Set project
[-split (no)]     Split memory array with centerline row decoding
[-rules (no)]     Evaluate process rules
[-eval (no)]      Evaluate cell library
[-spice (no)]     Generate spice file
[-gds (no)]       Generate GDSII file
[-help]           Display command line options


-------------- CGaAs SRAM Layout Extractor ------------
usage: ram_extract <options>
[-help]                Display command line options
[-eval (no)]           Evaluate cell library
[-ramonly (no)]        Evaluate and extract RAM cells only
[-buffonly (no)]       Evaluate and extract buffer cells only
[-num cell_number]     Evaluate all cells, but extract only one cell
  1 = 6T memory cell
  2 = Sense amplifier
  3 = Write buffer
  4 = Precharger
  5 = Word buffer
  6 = Row decoder


-------- CGaAs 6T RAM Cell Stability Analyzer ---------
usage: ram_stability -wu <wu> -wd <wd> -wp <wp>
-wu                          Width of pullup transistors (nm)
-wd                          Width of pulldown transistors (nm)
-wp                          Width of pass transistors (nm)
[-lp (500)]                  Length of pass transistors (nm)
[-m name (spice.models)]     Device models file
[-skew skew(30)]             Amount of skew in layout variation
[-help]                      Display command line options
```

```
---------- CGaAs RAM SPICE File Generator ------------
usage: ram_spice -row <rows> -col <cols> -T <period> filename
-row                          Number of rows
-col                          Number of columns
-T                            Cycle time (ps)
[-x (8)]                      # of cells between horizontal power rails
[-y (8)]                      # of cells between vertical power rails
[-c name (ramcells.spice)]    RAM cell library include filename
[-m name (spice.models)]      Device models file
[-dp (30)]                    Simulation datapoints per cycle
[-rf (300)]                   Clock rise/fall time (ps)
[-vdd (1300)]                 VDD value (mV)
[-split (no)]                 Split memory array - centerline row decoding
[-exe (no)]                   Execute simulation with HSPICE
[-ftype (0)]                  Spice file type (0-6)
[-r (project.go)]             Set process ruleset
[-p (project.go)]             Set project
[-help]                       Display command line options


----------------- CGaAs RAM Optimizer -----------------
usage: ram_optimize -row <rows> -col <cols> -d1 <decode1>
-row                          Number of rows
-col                          Number of columns
-d1                           Primary column decoding ratio (1,2,4,8,16)
[-d2 (0)]                     Secondary column decoding ratio (0,1,2,4)
[-split (no)]                 Split memory array - centerline row decoding
[-length (500)]               Gate length (nm)
[-m name (spice.models)]      Device models file
[-dp (30)]                    Simulation datapoints per cycle
[-rf (300)]                   Clock rise/fall time (ps)
[-vdd (1300)]                 VDD value (mV)
[-rules (no)]                 Evaluate process rules ONLY
[-help]                       Display command line options


---------------- CGaAs RAM Rail Sizer -----------------
usage: ram_railsize -row <rows> -col <cols> -f <frequency> -d1 <decode1>
-row                            Number of rows
-col                            Number of columns
-f                              Operating frequency (MHz)
-d1                             Primary column decoding ratio (1,2,4,8,16)
[-d2 (0)]                       Secondary column decoding ratio (0,1,2,4)
[-split (no)]                   Split memory array - centered row decoding
[-length (500)]                 Gate length (nm)
[-d (1000)]                     Maximum current density (uA per micron)
[-mv (250)]                     Maximum voltage drop (mV)
[-trace (optimize_RAM.trace)]   Optimization trace filename
[-m name (spice.models)]        Device models file
[-dp (30)]                      Simulation datapoints per cycle
[-rf (300)]                     Clock rise/fall time (ps)
[-vdd (1300)]                   VDD value (mV)
[-help]                         Display command line options
```

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Gwennap, Linley, "CPU Vendors Deploy Half-Micron Processes," *Microprocessor Report*, vol. 8., no. 5, April 18, 1994, pp. 16-20.

[2] Gwennap, Linley, "Microprocessors Lead the Way to 0.35 Microns," *Microprocessor Report*, vol. 9, no. 9, July 10, 1995, pp. 16-20.

[3] Gwennap, Linley, "IC Vendors Prepare for 0.25-Micron Leap," *Microprocessor Report*, vol. 10, no. 12, September 16, 1996, pp. 11-15.

[4] R. Dennard et al., *Semiconductor Silicon Electrochemical Society*, (H. Huff and R. Burgess, eds.), 1973.

[5] R. Dennard, F. Gaensslen, H. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE Journal of Solid State Circuits*, vol. SC-9, no. 5, October 1974, pp. 256-267.

[6] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Deisign*. Boston: Addison Wesley, 1993.

[7] A. Shojiro and Y. Wada, "Technology Challenges for Integration Near and Below 0.1 μm," *Proceedings of the IEEE*, vol. 85, no. 4, April 1997, pp. 505-520.

[8] W. Maly, H. Heineken, J. Khare, and P.K. Nag, "Design for Manufacturability in Submicron Domain," *Proceedings of the 1996 IEEE International Conference on Computer Aided Design*, pp. 690-697.

[9] G. Moore, "Cramming more components onto integrated circuits," Electronics, vol. 38, no. 8, April 19, 1965, pp. 114-117.

[10] G. Hutcheson and J. Hutcheson, "Technology and Economics in the Semiconductor Industry," *Scientific American*, vol. 274, no. 1, January 1996, pp. 54-62.

[11] J. Hutcheson and G. Hutcheson, "Is Semiconductor Manufacturing Equipment Still Affordable?" *Proceedings of the 1993 IEEE International Symposium on Semiconductor Manufacturing*

[12] D. Hicks and S. Brown, "Affordability Concerns in Advanced Semiconductor Manufacturing: The Nature of Industrial Linkage," *Proceedings of the 1995 IEEE International Symposium on Semiconductor Manufacturing*.

[13] W. Maly, "Computer-Aided Design for VLSI Circuit Manufacturability," *Proceedings of the IEEE*, vol. 78, no. 2, February 1990, pp. 356-392.

[14] A. Strojwas, "Design for Manufacturability and Yield," *Proceedings of the 1989 ACM/IEEE Design Automation Conference*, pp. 454-459.

[15] S. Director, W. Maly, and A. Strojwas, *VLSI Design for Manufacturing: Yield Enhancement*. Boston: Kluwer Academic Publishers, 1988.

[16] W. Maly, H. Heineken, and F. Agricola, "Yield Model for Manufacturing Strategy Planning and Product Shrink Applications," *Semiconductor International*, July 1994, pp. 148-154.

[17] H. Heineken, J. Khare, and W. Maly, "Yield Loss Forecasting in the Early Phases of the VLSI Design Process," *Proceedings of the 1996 IEEE Custom Integrated Circuits Conference*, pp. 27-30.

[18] W. Maly and J. Deszczka, "Yield Estimation Model for VLSI Artwork Evaluation," *Electron Letters*, vol. 19, no. 6, March 1983, pp. 226-227.

[19] W. Maly, "Modeling of Lithography-Related Yield Losses for CAD of VLSI Circuits," *IEEE Transactions on Computer Aided Design*, vol. 4, no. 4, July 1985, pp. 166-177.

[20] A. Ferris-Prabhu, "Role of Defect Size Distribution in Yield Modeling," *IEEE Transactions on Electron Devices*, vol. 32, no. 9, September 1985, pp. 1727-1736.

[21] H. Heiniken and W. Maly, "Interconnect Yield Model for Manufacturability Prediction in Synthesis of Standard Cell Based Designs," *Proceedings of the 1996 ACM/IEEE International Conference on Computer-Aided Design*, pp. 368-373.

[22] P. Nag, "Yield Forecasting," Ph.D. Thesis, Carnegie Mellon University, April 1996.

[23] R. Kessler, R. Jooss, A. Lebeck, and M. Hill, "Inexpensive Implementations of Set-Associativity," *Proceedings of the 1989 ACM/IEEE International Symposium on Computer Architecture*, pp. 131-138.

[24] A. Seznec, "A Case for Two-way Skewed-associative Caches," *Proceedings of the 1993 ACM/IEEE International Symposium on Computer Architecture*, pp. 169-178.

[25] A. Agarwal and S. Pudar, "Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches," *Proceedings of the 1993 ACM/IEEE International Symposium on Computer Architecture*, pp. 179-190.

[26] K. Theobald, H. Hum, and G. Gao, "A Design Framework for Hybrid-Access Caches," *Proceedings of the 1995 IEEE Symposium on High-Performance Computer Architecture*, pp. 144-153.

[27] B. Calder, D. Grunwald, and J. Emer, "Predictive Sequential Associative Cache," *Proceedings of the 1996 IEEE Symposium on High-Performance Computer Architecture*, pp. 244-253.

[28] S. Przybylski, M. Horowitz, and J. Hennessy, "Performance Tradeoffs in Cache Design," *Proceedings of the 1988 ACM/IEEE International Symposium on Computer Architecture*, pp. 290-298.

[29] R. Heald, K. Shin, V. Reddy, I. Kao, M. Khan, W. Lynch, G. Lauterbauch, J. Petolino, "64kB Sum-Addressed-Memory Cache with 1.6ns Cycle and 2.6ns Latency," Proceedings of the 1998 IEEE International Solid State Circuits Conference, pp. 350-351.

[30] J.K. Abrokwah, J.H. Huang, W. Ooms, C. Shurboff, J.A. Hallmark, R. Lucero, "A Manufacturable Complementary GaAs Process," *Proceedings of the 1993 IEEE GaAs IC Symposium*, pp. 127-129.

[31] B. Bernhardt, *et. al.*, "Complementary GaAs (CGaAs^{TM}): A High Performance BiCMOS Alternative," *Proceedings of the 1995 IEEE GaAs IC Symposium*, pp. 18-21, 1995.

[32] R. Brown, B. Bernhardt, M. LaMacchia, J. Abrokwah, P. Parakh, T. Basso, S. Gold, S. Stetson, C. Gauthier, D. Foster, B. Crawforth, T. McQuire, K. Sakallah, R. Lomax, and T. Mudge, "Overview of Complementary GaAs Technology for High-Speed VLSI Systems," *IEEE Transactions on VLSI Systems*, March 1998, vol. 6, no. 1, pp. 47-51.

[33] J. Abrokwah, R. Lucero, J. Hallmark, and B. Bernhardt, "Submicron P-Channel (Al,Ga)As/(In,Ga)As HIGFET's," *IEEE Transactions on Electron Devices*, vol. 44, no. 7, July 1997, pp. 1040-1045.

[34] R. Brown, T. Basso, P. Parakh, S. Gold, C. Gauthier, R. Lomax, T. Mudge, "Complementary GaAs Technology for a GHz Microprocessor," *Proceedings of the 1996 IEEE GaAs IC Symposium*, pp. 313-316.

[35] B. Tuck, "Taking Different Approaches to Design Reuse," *Computer Design Technology and Design Directions*, February 1998.

[36] C. Ajluni, "Cell-Library Development Tool Eases Migration to Next-Generation ICs," *Electronic Design Automation*, v. 45, no. 5, March 3, 1997, pp. 77-78.

[37] J. Mohoney, "FPGAs Shrink with Physical Design Reuse," *Integrated System Design*, April 1997.

[38] K. Rousseau, "Advanced Processes Put Pressure on Library Development," *Electronic Design*, vol. 43, no. 7, April 3, 1995, pp. 68-70.

[39] J. McLeod, "The State of the Art in IC Layout Migration," *Integrated System Design*, June, 1995.

[40] C. Stapper, R. Amstrong, and K. Saji, "Integrated Circuit Yield Statistics," Proceedings of the IEEE, vol. 71, April 1983, pp. 453-470.

[41] C. Stapper, "Modeling of Integrated Circuit Defect Sensitivities," *IBM Journal of Research and Development*, vol. 27, June 1983, pp. 549-557.

[42] W. Maly, M. Thomas, J. Chinn, and D. Campbell, "Characterization of Type, Size and Density of Spot Defects in Metallization Layer," *Yield Modeling and Fault Tolerance in VLSI*, 1988, pp. 72-90.

[43] W. Maly, "Cost of Silicon Viewed from VLSI Design Perspective," *Proceedings of the 1994 ACM/IEEE Design Automation Conference*, pp. 135-142.

[44] C. Barrett, "Microprocessor Evolution and Technology Impact," *Proceedings of the 1993 Symposium on VLSI Technology*, May 17-19, Kyoto Japan, pp. 7-10.

[45] S. Pulliam and D. Takahashi, "Some analysts praise Intel's prospects anew, depsite haevy pressure on its profit margins", *Wall Street Journal* (Eastern Edition): C4, February 4, 1998.

[46] W. Wade, "Behind Micron's Profit Slump," *Electronic News*, vol 44. no. 58, January 12, 1998.

[47] S. Pulliam, "Intel's solid earnings fail to offset other worries," *Wall Street Journal* (Easter Edition): C1, April 15, 1997.

[48] "Gizmos, yes. Profits, maybe." *Business Week*, no. 107, Jan 13, 1997.

[49] G. Hill, "Intel's earnings surged by 41% in 3rd quarter," *Wall Street Journal* (Eastern Edition): A3, October 17, 1995.

[50] J. Epstein, "Why Andy Grove should be worried: Intel looks stronger than ever, but its competitive advantages are slipping away," *Financial World*, no. 164, Aug 1, 1995, pp. 28-31.

[51] R. Razdan and A. Strojwas, "A Statistical Design Rule Developer," *IEEE Transactions on Computer Aided Design*, vol. CAD-5, no. 4, October 1986, pp. 508-520.

[52] I. Chen and A. Strojwas, "Realistic Yield Simulation for VLSIC Structural Failures," *IEEE Transactions on Computer Aided Design*, vol. CAD-6, no. 6, November 1987, pp. 965-980.

[53] A. Ferris-Prabhu, "Modeling of Critical Area in Yield Forecasts," *IEEE Journal of Solid State Circuits*, vol. SC-20, August 1985, pp. 874-875.

[54] A. Ferris-Prabhu, "Defect Size Variations and their Effect on the Critical Area of VLSI Devices," *IEEE Journal of Solid State Circuits*, vol. SC-20, August 1985, pp. 878-880.

[55] W. Maly, "Modeling of Point Defect Related Yield Losses for CAD of VLSI Circuits," *Proceedings of the IEEE 1984 International Conference on Computer Aided Design*, pp. 161-163.

[56] J. Khare, W. Maly, S. Griep, and D. Schmitt-Landsiedel, "Yield-Oriented Computer Aided Defect Diagnosis," *IEEE Transactions on Semiconductor Manufacturing*, vol. 8, no. 2, May 1995, pp. 195-205.

[57] W. Maly, "Future of Testing: Reintegration of Design, Testing, and Manufacturing," *Proceedings of the 1996 European Design and Test Conference.*

[58] A. Ipri, "Impact of Design Rule Reduction on Size, Yield, and Cost of Integrated Circuits," *Solid State Technology*, vol. 22, no. 2, February 1979, pp. 85-91.

[59] R. Rung, "Determining IC Layout Rules for Cost Minimization," *IEEE Journal of Solid-State Circuits*, vol. SC-16, no. 1, February 1981, pp. 35-43.

[60] G. Allen, A. Walton, and R. Holwill, "An Yield Improvement Technique for IC Layout Using Local Design Rules," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 11, November 1992, pp. 1355-1362.

[61] L. Gwennap, "Estimating IC Manufacturing Costs," *Microprocessor Report*, vol. 7, no. 10, August 2, 1993, pp. 12-16.

[62] B. Murphy, "Cost-Size Optima of Monolithic Integrated Circuits," *Proceedings of the IEEE*, vol. 52, 1964, pp. 1537-1545.

[63] R. Seeds, "Yield, Economic and Logistical Models for Complex Digital Arrays," *IEEE International Convention Record*, part 6, 1967, pp. 60-61.

[64] A. Dingwall, "High-Yield-Processed Bipolar LSI Arrays," *IEDM Technical Digest*, 1968.

[65] T. Okabe, M. Nagate, and S. Shimada, "Analysis of Integrated Circuits and New Expression for the Yield," *Electrical Engineering Japan*, vol. 92, pp. 135-141, 1972.

[66] R. Warner, "Applying a Composite Model to the Yield Problem," *IEEE Journal of Solid-State Circuits*, vol. SC-9, 1974, pp. 96-103.

[67] C. Stapper, "On a Composite Model to the IC Yield Problem," *IEEE Journal of Solid-State Circuits*, vol. SC-10, 1975, pp. 537-539.

[68] C. Stapper, "LSI Yield and Process Monitoring," *IBM Journal of Research and Development*, vol. 20, 1976, pp. 228-234.

[69] O. Paz and T. Lawson, "Modifications of Poisson Statistics: Modeling Defects Induced by Diffusion," *IEEE Journal of Solid-State Circuits*, vol. SC-12, 1977, pp. 540-546.

[70] S. Hu, "Some Considerations in the Formulation of IC Yield Statistics," *Solid-State Electronics*, vol. 22, 1979, pp. 205-211.

[71] C. Kooperberg, "Circuit Layout and Yield," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 4, August 1988, pp. 887-892.

[72] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann: San Mateo, 1990.

[73] K. Hedlund, "Models and Algorithms for Transistor Sizing in MOS circuits," *Proceedings of the 1984 IEEE/ACM International Conference on Computer-Aided Design*, pp. 12-14.

[74] B. Richman, J. Hansen, and K. Cameron, "A Deterministic Algorithm for Automatic CMOS Transistor Sizing," *Proceedings of the 1987 IEEE Custom Integrated Circuits Conference*, pp. 421-424.

[75] Z. Dai and K. Asada, "MOSIZ: A Two-step Transistor Sizing Algorithm based on Optimal Timing Assignment Method for Multi-stage Complex Gates," *Proceedings of the 1989 IEEE Custom Integrated Circuits Conference*, pp. 17.3.1 - 17.3.4

[76] B. Hoppe, G. Neuendorf, and D. Schmitt-Landsidel, "Automatic Transistor Sizing in High Performance CMOS Logic Circuits," *Proceedings of the 1989 European Computer Conference*, pp. 5-25 - 5-27.

[77] S. Sapatnekar, V. Rao, and P. Vaidya, "A Convex Optimization Approach to Transistor Sizing for CMOS Circuits," *Proceedings of the 1991 IEEE/ACM International Conference on Computer-Aided Design*, pp. 482-485.

[78] L. Heusler and W. Fichtner, "Transistor Sizing for Large Combinational Digital CMOS Circuits," *Integration, The VLSI Journal*, vol. 10, no. 2, 2 Jan. 1991, pp. 155-168.

[79] S. Sapatnekar, V. Rao, and P. Vaidya, "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12. no. 11, November 1993, pp. 1621-1634.

[80] W. Kao, N. Fathi, and C. Lee, "Algorithms for Automatic Transistor Sizing in CMOS Digital Circuits," *Proceedings of the 1995 ACM/IEEE Design Automation Conference*, pp. 781-784.

[81] N. Azemard, V. Bonzom, and D. Auvergne, "P.SIZE: A Sizing Aid for Optimized Designs," *Proceedings of the 1992 European Design Automation Conference*, pp. 160-165.

[82] U. Ko and P. Balsara, "Short-Circuit Power Driven Gate Sizing Technique for Reducing Power Dissipation," *IEEE Transactions on VLSI Systems*, vol. 3, no. 3, September 1995, pp. 450-455.

[83] J. Shyu, J. Fishburn, A. Dunlop, and A. Sangiovanni-Vincentelli, "Optimization-based Transistor Sizing," *Proceedings of the 1987 IEEE Custom Integrated Circuits Conference*, pp. 417-420.

[84] G. Chen, H. Onodera, and K. Tamaru, "An Iterative Gate Sizing Approach with Accurate Delay Evaluation," *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, pp. 422-427.

[85] S. Mehrotra, P. Franzon, and W. Liu, "Stochastic Optimization Approach to Transistor Sizing for CMOS VLSI Circuits," *Proceedings of the 1994 ACM/IEEE Design Automation Conference*, pp. 36-40.

[86] W. Nye, D. Riley, A. Sangiovanni-Vincentelli, and A. Tits, "DELIGHT.SPICE: An Optimization-based System for the Design of Integrated Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 7 no. 4, April 1988, pp. 501-519.

[87] J. Fishburn and A. Dunlop, "TILOS: A Posynomial Programming Approach to Transistor Sizing," *Proceedings of the 1985 IEEE International Conference on Computer-Aided Design*, pp. 326-328.

[88] S. Sapatnekar and V. Rao, "iDEAS: A Delay Estimator and Transistor Sizing Tool for CMOS Circuits," *Proceedings of the 1990 IEEE Custom Integrated Circuits Conference*, pp. 9.3.1 - 9.3.4.

[89] K. Hedlund, "Aesop: A Tool for Automated Transistor Sizing," *Proceedings of the 1987 ACM/IEEE Design Automation Conference*, pp. 114 -120.

[90] M. Cirit, "Transistor Sizing in CMOS Circuits," *Proceedings of the 1987 ACM/ IEEE Design Automation Conference*, pp. 121-124.

[91] A. Conn, P. Coulman, R. Haring, G. Morrill, and C. Visweswariah, "Optimization of Custom MOS Circuits by Transistor Sizing," *Proceedings of the 1996 IEEE/ ACM International Conference on Computer-Aided Design*, pp. 174-180.

[92] D. Chen and C. Zukowski, "CMOS Optimization Including Logic Family Mixing," *Proceedings of the 1991 IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 2240-2243.

[93] H. Chen and S. Kang, "A New Circuit Optimization Technique for High Performance CMOS Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 5, May 1991, pp. 670-676.

[94] J. Kao, A. Chandrakasan, and D. Antoniadis, "Transistor Sizing Issues and Tool for Multi-threshold CMOS Technology," *Proceedings of the 1997 ACM/IEEE Design Automation Conference*, pp. 409-414.

[95] D. Marple, "Transistor Size Optimization in the Tailor Layout System," *Proceedings of the 1989 ACM/IEEE Design Automation Conference*, pp. 43-48.

[96] M. Yamada, S. Kurosawa, R. Nojima, N. Kojima, T. Mitsuhashi, and N. Goto, "Synergistic Power/Area Optimization with Transistor Sizing and Wire Length Minimization," *IEICE Transactions on Electronics*, vol. E78-C, no. 4, April 1995, pp. 441-445.

[97] N. Menezes, R. Baldick and L. Pileggi, "A Sequential Quadratic Programming Approach to Concurrent Gate and Wire Sizing," *Proceedings of the 1995 IEEE/ ACM International Conference on Computer-Aided Design*, pp. 144-151.

[98] J. Cong and L. He, "An Efficient Approach to Simultaneous Transistor and Interconnect Sizing," *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, pp. 181-186.

[99] W. Swartz, C. Giuffre, W. Banzhaf, M. deWit, H. Khan, C. McIntosh, T. Pavey, and D. Thomas, "CMOS RAM, ROM, and PLA Generators for ASIC Applications," *Proceedings of the 1986 IEEE Custom Integrated Circuits Conference*, pp. 334 - 338.

[100] J. Drummond and M. Lepkowski, "BiCMOS Submicron Compiler Memories," *Proceedings of the 1990 IEEE ASIC Seminar and Exhibit*, pp. 3-3.1 - 3-3.3.

[101] T. Dao and F. Svejda, "A Dual-port SRAM Compiler for 0.8µm 100K BiCMOS Gate Arrays," *Proceedings of the 1991 IEEE Custom Integrated Circuits Conference*, pp. 22.4.1-22.4.3.

[102] T. Le, H. Phuong, and P. Lin, "1KX128 High-performance, Low-power configurable CMOS SRAM Compiler," *Proceedings of the 1990 IEEE ASIC Seminar and Exhibit*, pp. 3-5.1 - 3-5.4.

[103] K. Tsao, N. Zhu, and T. Pham, "A High Performance Memory Compiler for Multi-port RAMs," *Proceedings of the 1990 IEEE ASIC Seminar and Exhibit*, pp. 3-6.1 - 3-6.4.

[104] H. Shinohara, N. Matsumoto, K. Fujimori, Y. Tsujihashi, H. Nakao, S. Kato, Y. Horiba, and A. Tada, "A Flexible Multiport RAM Compiler for Data Path," *IEEE Journal of Solid State Circuits*, vol. 26, no. 3., March 1991, pp. 343-348.

[105] J. Tou, P. Gee, J. Duh, and R. Eesley, "A Sub-micron CMOS Embedded SRAM Compiler," *Proceedings of the 1991 IEEE Custom Integrated Circuits Conference*, pp. 22.3.1-22.3.4.

[106] A. Chandna, C. Kibler, R. Brown, M. Roberts, and K. Sakallah, "The Aurora RAM Compiler," *Proceedings of the 1995 ACM/IEEE Design Automation Conference*, pp. 261-266.

[107] A. Chandna, "GaAs MESFET Static RAM Design for Embedded Applications," Ph.D. Dissertation, University of Michigan, 1995.

[108] A. Chandna and R. Brown, "An Asynchronous GaAs MESFET Static RAM Using a New Current Mirror Memory Cell," *IEEE Journal of Solid State Circuits*, October 1994, pp. 1270-1276.

[109] L. Tavrow, J. Johnson, and M. Santoro, "Design Methodology for Quickly and Accurately Generating SRAMs," *Integrated System Design*, February 1998, pp. 24-36.

[110] D. Boyer, "Symbolic Layout Compaction Review," *Proceedings of the 1988 ACM/ IEEE Design Automation Conference*, pp. 383-389.

[111] C. Carpenter and M. Horowitz, "Generating Incremental VLSI Compaction Spacing Constraints," *Proceedings of the 1987 ACM/IEEE Design Automation Conference*, pp. 291-297.

[112] M. Silva and M. Lanca, "INTCOM - An Expert System for Compaction of IC Layouts," *Proceedings of the 1987 IEEE Mediterranean Electrotechnical Conference*, pp. 285-288.

[113] H. Shin and C. Lo, "An Efficient Two-Dimensional Layout Compaction Algorithm," *Proceedings of the 1989 ACM/IEEE Design Automation Conference*, pp. 290-295.

[114] W. Schiele, "Compaction with Incremental Over-Constraint Resolution," *Proceedings of the 1988 ACM/IEEE Design Automation Conference*, pp. 390-395.

[115] C. Lo and R. Varadarajan, "An $O(n^{1.5}\log n)$ 1-d Compaction Algorithm," *Proceedings of the 1990 ACM/IEEE Design Automation Conference*, pp. 382-387.

[116] D. Boyer, "Process Independent Constraint Graph Compaction," *Proceedings of the 1992 ACM/IEEE Design Automation Conference*, pp. 318-322.

[117] L. Nyland, "Improving Virtual-Grid Compaction Through Grouping," *Proceedings of the 1987 ACM/IEEE Design Automation Conference*, pp. 305-310.

[118] R. Byrne and G. Shoja, "Distributed System for VLSI Layout Compaction," *Proceedings of the IEEE*, vol. 141, no. 1, pp. 49-56.

[119] C. Kingsley, "A Hierarchical, Error-Tolerant Compactor," *Proceedings of the 1984 ACM/IEEE Design Automation Conference*, pp. 126-132.

[120] G. Entenman and S. Daniel, "A Fully Automatic Hierarchical Compactor," *Proceedings of the 1985 ACM/IEEE Design Automation Conference*, pp. 69-75.

[121] M. Reichelt and W. Wolf, "An Improved Cell Model for Hierarchical Constraint-Graph Compaction," *Proceedings of the International Conference on Computer Aided Design*, pp. 482-485, 1986.

[122] W. Crocker, R. Varadarajan, and C. Lo, "MACS: A Module Assembly and Compaction System," *Proceedings of the International Conference on Computer Design*, pp. 205-208, October, 1987.

[123] H. Shin, A. Sangiovanni-Vincentelli, and C. Sequin, "Two-Dimensional Module Compactor Based on 'Zone-Refining'," *Proceedings of the International Conference on Computer Design*, pp. 201-204, 1987.

[124] A. deLange, J. deLange, and J. Vink, "A Hierarchical Constraint Graph Generation and Compaction System for Symbolic Layout," *Proceedings of the International Conference on Computer Design*, pp. 532-535, 1989.

[125] D. Marple, "A Hierarchy Preserving Hierarchical Compactor," *Proceedings of the 1990 ACM/IEEE Design Automation Conference*, pp. 375-381.

[126] C. Bamji and R. Varadarajan, "Hierarchical Pitchmatching Compaction Using Minimum Design," *Proceedings of the 1992 ACM/IEEE Design Automation Conference*, pp. 311-317.

[127] W. Kim, J. Lee, and H. Shin, "A New Hierarchical Layout Compactor Using Simplified Graph Models," *Proceedings of the 1992 ACM/IEEE Design Automation Conference*, pp. 323-326.

[128] S. Yao, C. Cheng, D. Dutt, S. Nahar, and C. Lo, "Cell-Based Hierarchical Pitchmatching Compaction Using Minimal LP," *Proceedings of the 1993 ACM/ IEEE Design Automation Conference*, pp. 395-400.

[129] C. Bamji and R. Varadarajan, "MSTC: A Method for Identifying Overconstraints during Hierarchical Compaction," *Proceedings of the 1993 ACM/IEEE Design Automation Conference*, pp. 389-394.

[130] K. Okada, H. Onodera, and K. Tamaru, "Compaction with Shape Optimization," *Proceedings of the IEEE 1994 Custom Integrated Circuits Conference*, pp. 545-548.

[131] K. Okada, H. Onodera, and K. Tamaru, "Compaction with Shape Optimization and its Application to Layout Recycling," *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, vol. E78-A, no. 2, February, 1995, pp. 169-176.

[132] L. Wang, Y. Lai, B. Liu, and T. Chang, "Performance-directed Compaction for VLSI Symbolic Layouts," *Computer Aided Design*, vol. 27, no. 1, January 1995, pp. 65-74.

[133] P. Hsieh and V. Chang, "A Symbolic Layout Synthesis System for VLSI Design," *Proceedings of the IEEE 1987 Custom Integrated Circuits Conference*, pp. 507-511.

[134] B. Henderson, R. Kumar, R. Radojcic, and M. Bui, "Foundry Portable Designs," *Integrated Circuit Design*, April 1995.

[135] J. Dao, N. Matsumoto, T. Hamai, C. Ogawa, and S. Mori, "A Compaction Method for Full Chip VLSI Layouts," *Proceedings of the 1993 ACM/IEEE Design Automation Conference*, pp. 407-412.

[136] Z. Apanovich and A. Marchuk, "DECOMP: a Technology Migration Subsystem for Full Chip Mask Layouts," *Proceedings of the 1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 942-945.

[137] S. Kishida, Y. Shibayama, H. Tanizaki, A. Hanami, and I. Ohkura, "Transistor Size Optimization in Layout Design Rule Migration," *Proceedings of the IEEE 1994 Custom Integrated Circuits Conference*, pp. 541-544.

[138] D. Kinell and D. Wilson, "GaAs Static Random Access Memory Cell Design," *Research abstract of the 1990 IEEE GaAs IC Symposium.*

[139] J. Notthoff, R. Krein, J. Stephens, G. Troeger, C. Vogelsang, and C. Hyun, "A 4K X 1 Bit Complementary E-JFET Static RAM," *Proceedings of the 1987 IEEE GaAs IC Symposium*, pp. 185-188.

[140] C. Vogelsang, J. Castro, J. Notthoff, G. Troeger, J. Stephens, and R. Krein, "Complementary GaAs JFET 16K SRAM," *Proceedings of the 1988 IEEE GaAs IC Symposium*, pp. 75-78.

[141] M. Suzuki, S. Notomi, M. Ono, N. Kobayashi, E. Mitani, K. Odani, T. Mimura, and M. Abe, "A 1.2ns HEMT 64kb SRAM," *Proceedings of the 1991 IEEE International Solid-State Circuits Conference*, pp. 48-49.

[142] J. Hallmark, C. Shurboff, B. Oohms, R. Lucero, J. Abrokwah, and J. Huang, "0.9 V DSP Blocks: A 15 ns 4 K SRAM and a 45 ns 16-bit Multiply/Accumulator," *Proceedings of the 1994 IEEE GaAs IC Symposium*, pp. 55-58.

[143] E. Seevinck, P. van Beers, and H. Ontrop, "Current-mode Techniques for High Speed VLSI Circuits with Application to Current Sense Amplifier for CMOS SRAMs," *IEEE Journal of Solid State Circuits*, vol. 26, no. 4, April 1991, pp. 525-535.

[144] H. Nambu, K. Kanetani, Y. Idei, N. Homma, K. Yamaguchi, T. Hiramoto, N. Tamba, M. Odaka, K. Watanabe, T. Ikeda, K. Ohhata, and Y. Sakurai, "High-speed Sensing Techniques for Ultrahigh-speed SRAMs," *IEEE Journal of Solid State Circuits*, vol. 27, no. 4, April 1992, pp. 632-639.

[145] T. Yabe, "High-speed Circuit Techniques for 1 to 5 V Operating Memories," *IEICE Transactions on Electronics*, vol. E76-C, no. 5, May 1993, pp. 708-713.

[146] K. O'Connor, "A Source Sensing Technique Applied to SRAM Cells," *IEEE Journal of Solid State Circuits*, vol. 30, no. 4, April 1995, pp. 500-511.

[147] K. Ishibashi, "High-Speed CMOS SRAM Technologies for Cache Applications," *IEICE Transactions on Electronics*, vol. E79-C, no. 6, June 1996, pp. 724-734.

[148] K. Ohhata, T. Kunsunoki, H. Nambu, K. Kanetani, T. Masuda, M. Ohayashi, S. Hamamoto, K. Yamaguchi, Y. Idei, and N. Homma, "Redundancy Circuit for a Sub-nanosecond, Megabit ECL-CMOS SRAM," *IEICE Transactions on Electronics*, vol. E79-C, no. 3, March 1996, pp. 415-422.

[149] A, Suzuki, T. Kobayashi, T. Hamano, H. Hatada, A. Kawasumi, F. Matsuoka, K. Ishimaru, M. Takahashi, M. Nishigohri, Y. Okayama, Y. Unno, M. Kakumu, and J. Tsujimoto, "A 400MHz 4.5Mb Synchronous BiCMOS SRAM with Alternating Bit-line Loads," *Proceedings of the 1996 IEEE International Solid-state Circuits Conference*, pp. 146-147.

[150] *Grafedit User's Guide and Reference Manual*, Cascade Design Automation, 1996.

[151] C. Fisher, R. Blankenship, J. Jensen, T. Rossman, and K. Svilich, "Optimization of Standard Cell Libraries for Low Power, High Speed, or Minimal Area Designs," *Proceedings of the 1996 IEEE Custom Integrated Circuits Conference*, pp. 493-496.

[152] B. Bateman, "High Speed SRAM Design," Tutorial, *1998 IEEE International Solid State Circuits Conference.*

[153] E. Seevinck, F. List, and J. Lohstroh, "Static-Noise Margin Analysis of MOS SRAM Cells," *IEEE Journal of Solid State Circuits*, vol. SC-22, no. 5, October 1987, pp. 748-754.

[154] B. Chappell, S. Schuster, and G. Sai-Halasz, "Stability and SER Analysis of Static RAM Cells," *IEEE Journal of Solid State Circuits*, vol. SC-20, no. 1, February 1985, pp. 383-390.

[155] K. Anami, M. Yoshimoto, H. Shinohara, Y. Hirata, and T. Nakano, "Design Considerations of a Static Memory Cell," *IEEE Journal of Solid State Circuits*, vol. SC-18, no. 4, August 1983, pp. 414-417.

[156] L. Tomasetta, "GaAs Manufacturing: Myth and Reality," invited speaker, 1998 International Conference of GaAs Manufacturing Technology.

[157] Conversation with D. Roulstone, Ph.D. Candidate in Accounting, University of Michigan, November 23, 1998.

[158] Conference call with Carlos Gutierrez, Gian Gerosa, and Arup Bhattacharya of Motorola, Inc., Somerset Design Center, Austin, Texas, November 29, 1998.